Analysis of (1+1) Evolutionary Algorithm and Randomized Local Search with Memory

Chi Wan Sung albert.sung@cityu.edu.hk
Department of Electronic Engineering, City University of Hong Kong, Hong Kong

Shiu Yin Yuen kelviny.ee@cityu.edu.hk
Department of Electronic Engineering, City University of Hong Kong, Hong Kong

Abstract

This paper considers the scenario of the (1+1) evolutionary algorithm (EA) and randomized local search (RLS) with memory. Previously explored solutions are stored in memory until an improvement in fitness is obtained; then the stored information is discarded. This results in two new algorithms: (1+1) EA-m (with a raw list and hash table option) and RLS-m+ (and RLS-m if the function is a priori known to be unimodal). These two algorithms can be regarded as very simple forms of tabu search. Rigorous theoretical analysis of the expected time to find the globally optimal solutions for these algorithms is conducted for both unimodal and multimodal functions. A unified mathematical framework, involving the new concept of spatially invariant neighborhood, is proposed. Under this framework, both (1+1) EA with standard uniform mutation and RLS can be considered as particular instances and in the most general cases, all functions can be considered to be unimodal. Under this framework, it is found that for unimodal functions, the improvement by memory assistance is always positive but at most by one half. For multimodal functions, the improvement is significant; for functions with gaps and another hard function, the order of growth is reduced; for at least one example function, the order can change from exponential to polynomial. Empirical results, with a reasonable fitness evaluation time assumption, verify that (1+1) EA-m and RLS-m+ are superior to their conventional counterparts. Both new algorithms are promising for use in a memetic algorithm. In particular, RLS-m+ makes the previously impractical RLS practical, and surprisingly, does not require any extra memory in actual implementation.

Keywords

(1+1) EA, randomized local search, expected optimization time, memory, tabu search.

1 Introduction

Randomized search algorithms have been very successful in solving combinatorial optimization problems. They are particularly useful when little domain knowledge is at hand for tailoring algorithms. Well known examples include genetic algorithm (GA; Holland, 1975) and simulated annealing (Kirkpatrick et al., 1983). While these techniques have been proven effective in solving real-world problems, their performance, limitations, and relative merits are not well understood. To gain more insight, various

The work described in this paper was partially supported by a grant from the City University of Hong Kong (Project No. 7002603).

research efforts have been made. For example, there are some studies on whether a randomized search algorithm will ever converge to an optimal solution in a given optimization problem, provided that the runtime is long enough (e.g., Hajek, 1988; Rudolph, 1997). If the convergence of an algorithm is guaranteed, it is important to know how long it will take to reach an optimal point. Our work in this paper belongs to this line of study.

To study the runtime behavior, analyses on simple evolutionary algorithms (EAs) have been made. In particular, the (1+1) EA, which iterates a single point in the search space by mutation, is widely studied. Mühlenbein showed that the (1+1) EA can maximize the pseudo-Boolean function ONEMAX with an expected number of $O(n \log n)$ iterations (Mühlenbein, 1992). Garnier et al. (1999) generalized the result and derived the probability distribution of its runtime. Rudolph (1996), and Garnier and Kallel (2000) showed that the (1+1) EA has polynomial expected runtime on some functions having long paths. Droste et al. (2002a) proved a complementary result by showing that the expected runtime can be exponential for some long-path functions. A $(1 + \lambda)$ EA was considered in Jansen et al. (2005). The case where population size is larger than one was considered in He and Yao (2002, 2003) and in Witt (2006). The effect of the crossover operator was analyzed elsewhere (Jansen and Wegener, 2002a; Dietzfelbinger et al., 2003; Sudholt, 2005). Furthermore, analysis of the (1+1) EA on some combinatorial optimization problems can be found (Neumann and Wegener, 2007; Neumann, 2008; Oliveto et al., 2008). A recent survey on major analysis tools is given in Oliveto et al. (2007).

Apart from (1+1) EA, some other algorithms have been analyzed in the literature. Examples include simulated annealing (Jansen and Wegener, 2007), randomized local search (RLS: Sudholt, 2006), and the (1+1) memetic algorithm (Sudholt, 2006). In this paper, we consider the use of *memory* as an add-on to existing randomized search algorithms. By storing information on visited points in memory, re-evaluating those points can be avoided, thus reducing the number of fitness evaluations. This gain can be enormous for applications that involve expensive or time-consuming simulations to evaluate the fitness of a point. Examples can be found in works by Fong et al. (2008, 2010), which use simulation programs to evaluate the design of a heating, ventilating, and air-conditioning (HVAC) system. Clearly, using simulation to evaluate the fitness of a particular parameter setting is time-consuming, and it is commented that such a simulation-optimization approach has increasingly been applied to the study of HVAC systems (see Fong et al., 2008 and references therein). For those applications, storing visited points becomes necessary, and this idea has been used in Yuen and Chow (2008a, 2009) to design a nonrevisiting genetic algorithm, in which all visited points are stored in binary space partitioning tree. The same idea can be applied to other evolutionary algorithms as well (Yuen and Chow, 2008b; Chow and Yuen, 2008). In this paper, we store visited points in memory until fitness improvement has been made and then discard them. This design not only saves memory but also simplifies our analysis. To analyze the potential gain of our approach, we propose variants of the (1+1) EA and RLS, denoted by (1+1) EA-m and RLS-m, respectively, where the notation m stands for memory. Their runtimes on some pseudo-Boolean functions are derived.

The visited points stored in memory can be used not only to avoid revisits, but also to help generate new search points. To that end, we propose an extended version of RLS-m, denoted by RLS-m+. The idea is to adjust the number of flipped bits adaptively, so that searching can proceed from nearest neighbors to further ones in order to escape from local maxima. Expected runtimes of RLS-m+ on some pseudo-Boolean functions have been obtained and compared with RLS and the (1+1) EA. Our approach is reminiscent

of tabu search (Glover and Laguna, 1997). Indeed, both RLS-m and RLS-m+ can be regarded as tabu search strategies with short-term memory. On the other hand, these two algorithms use memory in a much simpler way than most tabu search strategies, and therefore are much more amenable to analysis. To the best of our knowledge, our work is the first to analyze expected runtimes of evolutionary algorithms with memory.

To generalize our results, we construct a unified framework for the class of (1+1) EAs with spatially invariant neighborhood structure. A lower bound of the relative performance of memory-assisted EAs is derived. An example is used to illustrate the use of the framework.

Empirical studies and implementations are also reported. We propose the use of hashing to store visited points for (1+1) EA. Numerical experiments show that this method is efficient for applications involving expensive fitness evaluation. For RLS-m+, we propose a method based on the idea of a random number generator, that surprisingly, involves no memory overhead.

The rest of this paper is organized as follows. In Section 2, we introduce some definitions and terminology. In Section 3, we describe three memory-assisted algorithms. In Section 4, we compare the expected runtimes of different algorithms on unimodal functions. In Section 5, we show that memory-assisted schemes can reduce the order of growth of expected runtime on some functions when compared with the (1+1) EA. In Section 6, we present a new framework to unify our results. In Section 7, we discuss how visited points can be stored and retrieved from memory. In Section 8, numerical results are presented. In Section 9, we draw conclusions.

2 Algorithms and Basic Properties

In this paper, we restrict attention to fitness functions that are discrete and binary. A function $f : \{0,1\}^n \to \Re$ is called a *fitness* function. The (1+1) EA tries to find an $\boldsymbol{x}^* \in \{0,1\}^n$ that maximizes f using the following procedure.

The (1+1) EA

1. Choose an initial point, $\boldsymbol{x}^{(0)}$, uniformly over $\{0,1\}^n$. Let $t := 0$.
2. Produce a point $\boldsymbol{x}'$ from $\boldsymbol{x}^{(t)}$ by some unary genetic operator.
3. Compare the fitness values of $\boldsymbol{x}'$ and $\boldsymbol{x}^{(t)}$ using some replacement indicator function $\Lambda : \Re^2 \to \{0,1\}$. If $\Lambda(f(\boldsymbol{x}'), f(\boldsymbol{x}^{(t)})) = 1$, then let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}'$. Otherwise, let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}^{(t)}$.
4. Increase t by one and repeat step 2.

Let $\mathcal{P}(t)$ be the finite sequence of points $\{\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}$. Define the runtime T of the (1+1) EA on a fitness function f as follows:

$$T \triangleq \min\left\{t : \left(\arg\max_{\boldsymbol{x}\in\{0,1\}^n} f(x)\right) \cap \mathcal{P}(t) \neq \emptyset\right\}.$$

This is also the number of function evaluations before finding an optimal solution, $\boldsymbol{x}^*$, excluding the evaluation of the initial point.

Note that the algorithm is defined without any stopping rule. The reason is that we are interested in analyzing its runtime. In practice, the algorithm should be stopped

after running a certain number of times or when no improvement is made after a certain number of iterations.

For (1+1) EA, there are two representative genetic operators: *standard mutation* and *one-bit flip*.[1] For standard mutation, the point $\boldsymbol{x}'$ is generated from $\boldsymbol{x}^{(t)}$ by independently flipping the n bits of $\boldsymbol{x}^{(t)}$, each with probability $1/n$. For one-bit flip, the point $\boldsymbol{x}'$ is generated by randomly flipping one bit of $\boldsymbol{x}^{(t)}$, chosen uniformly among the n bits. We adopt the convention that standard mutation is assumed if the genetic operator is not explicitly specified. In other words, we call the (1+1) EA with standard mutation simply the (1+1) EA. When one-bit flip is assumed, we call the resultant algorithm RLS.

The (1+1) EA is fully specified once the replacement indicator function is defined. One may define it as follows:

$$\Lambda(a, b) = \begin{cases} 1 & \text{if } a \geq b, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Another common way to define the function is to change the above inequality sign to a strict inequality. We call the former definition type-1 and denote the operator by Λ_1, whereas the latter one is called type-2 and the operator is denoted by Λ_2. If the type-i definition is used ($i = 1, 2$), we denote the resultant algorithm by either the (1+1) EA_i or RLS_i, depending on which mutation operator is used. The runtime of these two schemes are denoted by T_{si} and T_{ri}, respectively, where $i = 1, 2$, s stands for standard mutation, and r stands for randomized local search.

It was shown in Droste et al. (2002a) that the order of growth of $E[T_{s1}]$ for some functions can be smaller than that of $E[T_{s2}]$. The opposite can also be true, as shown in Jansen and Wegener (2002b). Here we show that these two expectations are equal for a special class of fitness functions. First we introduce a concept called the function of unitation (Srinivas and Patnaik, 1996).

DEFINITION 1: *A function $f : \{0, 1\}^n \to \Re$ is said to be a unitation function if $f(\boldsymbol{x})$ depends only on the Hamming weight of $\boldsymbol{x}$. In addition, if $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ for all pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ whose Hamming weights are different, we say that f is an injective unitation function.*

THEOREM 1: *If the fitness function of a given problem is an injective unitation function, then the probability distributions of T_{s1} and T_{s2} are the same for that problem.*

PROOF: Let the current point be $\boldsymbol{x}$ and the newly generated point be $\boldsymbol{x}'$. Since the fitness function f is an injective unitation function, the (1+1) EA_1 and the (1+1) EA_2 behave differently only if the Hamming weights of $\boldsymbol{x}$ and $\boldsymbol{x}'$ are the same and $\boldsymbol{x} \neq \boldsymbol{x}'$. However, the residual runtimes of these two algorithms depend only on the Hamming weight of the current point. Hence, the expected runtimes are the same whether $\boldsymbol{x}$ is replaced by $\boldsymbol{x}'$ or not. □

A similar result can be proven for RLS. It turns out that the condition can be loosened to the class of stepwise functions.[2]

[1] In the literature, standard mutation is sometimes called global mutation, whereas one-bit flip is sometimes called local mutation.

[2] An injective unitation function must be stepwise.

DEFINITION 2: *A function $f : \{0,1\}^n \to \Re$ is said to be stepwise if $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ for all pairs of $\boldsymbol{x}$ and $\boldsymbol{y}$ whose Hamming distances differ by one.*

THEOREM 2: *If the fitness function of a given problem is a stepwise function, then the probability distributions of T_{r1} and T_{r2} are the same for that problem.*

PROOF: Let the current point be $\boldsymbol{x}$ and the newly generated point be $\boldsymbol{x}'$. Since f is stepwise and the Hamming distances between $\boldsymbol{x}$ and $\boldsymbol{x}'$ must differ by one, $f(\boldsymbol{x}) \neq f(\boldsymbol{x}')$. Hence, RLS_1 and RLS_2 behave in exactly the same way. □

In this paper, we sometimes drop the second subscript and simply write T_s or T_r, with the understanding that the derived result applies to both cases.

3 Memory-Assisted Enhancement

Under both the (1+1) EA and RLS, the offspring are randomly generated. It is likely for them to generate points that have been evaluated before, which is clearly useless. This redundancy can be removed by storing some or all of the visited points and avoiding reevaluating their fitness values should they be generated again. To illustrate the idea, we formally define the (1+1) EA with memory as follows. In contrast to memoryless algorithms defined in the previous section, for all memory-assisted algorithms to be described below, we assume that the type-2 replacement operator is used.

The (1+1) EA-m

1. Choose an initial point, $\boldsymbol{x}^{(0)}$, uniformly over $\{0,1\}^n$, and put it into an empty set $\mathcal{S}$. Let $t := 0$.
2. Produce a point $\boldsymbol{x}'$ from $\boldsymbol{x}^{(t)}$ by standard mutation. Repeat until $\boldsymbol{x}' \notin \mathcal{S}$. Put $\boldsymbol{x}'$ into $\mathcal{S}$.
3. If $\Lambda_2(f(\boldsymbol{x}'), f(\boldsymbol{x}^{(t)})) = 1$, then let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}'$, and remove all elements except $\boldsymbol{x}'$ from $\mathcal{S}$. Otherwise, let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}^{(t)}$.
4. Increase t by one and repeat step 2.

Note that visited points are stored in a set called $\mathcal{S}$. In our design, once a replacement occurs, the memory is cleared; only the current point is retained. We do not store all visited points because on the one hand, the memory usage can be reduced, and on the other, the underlying process becomes semi-Markovian and the expected runtime is more amenable to analysis.

We denote the runtime of the (1+1) EA-m by T_{sm}. It is clear that T_{sm} is always less than or equal to T_{s2} in a stochastic sense, since points that have been evaluated before will not be generated again. We write this relation as $T_{\text{sm}} \preceq T_{\text{s2}}$, which precisely means that

$$\Pr\{T_{\text{sm}} > c\} \leq \Pr\{T_{\text{s2}} > c\} \text{ for all } c \in \Re_+. \tag{2}$$

The same idea can be applied to RLS. Let $d(\boldsymbol{x}, \boldsymbol{y})$ be the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$, and $\mathcal{B}_k(\boldsymbol{x})$ be the k-ball of $\boldsymbol{x}$ defined by $\{\boldsymbol{y} : d(\boldsymbol{x}, \boldsymbol{y}) \leq k\}$. The algorithm is stated below.

The RLS-m Algorithm

1. Choose an initial point, $\boldsymbol{x}^{(0)}$, uniformly over $\{0,1\}^n$, and put it into an empty set $\mathcal{S}$. Let $t := 0$.
2. If $\mathcal{B}_1(\boldsymbol{x}^{(t)})\backslash\mathcal{S} \neq \emptyset$, then choose a point $\boldsymbol{x}'$ uniformly from $\mathcal{B}_1(\boldsymbol{x}^{(t)})\backslash\mathcal{S}$, and put it into $\mathcal{S}$. If no such point exists, let $\boldsymbol{x}' := \boldsymbol{x}^{(t)}$. [3]
3. If $\Lambda_2(f(\boldsymbol{x}'), f(\boldsymbol{x}^{(t)})) = 1$, then let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}'$, and remove all elements except $\boldsymbol{x}'$ from $\mathcal{S}$. Otherwise, let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}^{(t)}$.
4. Increase t by one and repeat step 2.

It is well known that one shortcoming of RLS is that it may get trapped in local maxima. So may RLS-m. However, as visited points are stored under RLS-m, the information can be used to prevent trapping. We extend RLS-m and define RLS-m+ as follows.

The RLS-m+ Algorithm

1. Choose an initial point, $\boldsymbol{x}^{(0)}$, uniformly over $\{0,1\}^n$, and put it into an empty set $\mathcal{S}$. Let $t := 0$ and $k := 1$.
2. If $\mathcal{B}_k(\boldsymbol{x}^{(t)})\backslash\mathcal{S} \neq \emptyset$, then choose a point $\boldsymbol{x}'$ uniformly from $\mathcal{B}_k(\boldsymbol{x}^{(t)})\backslash\mathcal{S}$, and put it into $\mathcal{S}$. Otherwise, increase k by one and repeat step 2.
3. If $\Lambda_2(f(\boldsymbol{x}'), f(\boldsymbol{x}^{(t)})) = 1$, then let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}'$, $k := 1$, and remove all elements except $\boldsymbol{x}'$ from $\mathcal{S}$. Otherwise, let $\boldsymbol{x}^{(t+1)} := \boldsymbol{x}^{(t)}$.
4. Increase t by one and repeat step 2.

Unlike RLS and RLS-m, RLS-m+ has the additional capability of escaping from local maxima. Whenever it has tried all points within the 1-ball of the current point, it will proceed to flipping two bits and the process will go on until it escapes from the local maximum.

The runtimes of RLS-m and RLS-m+ are defined with respect to $\mathcal{P}(t)$ as before, and are denoted by T_{rm} and $T_{\mathrm{rm+}}$, respectively. It is clear that $T_{\mathrm{rm+}} \preceq T_{\mathrm{rm}} \preceq T_{\mathrm{r2}}$.

4 Two Useful Lemmas

In this section, we present two lemmas that will be used in later sections.

LEMMA 1: *Assume that $\boldsymbol{y}$ is the only point that has a strictly larger fitness value than $\boldsymbol{x}$ does. The expected time for (1+1) EA-m to reach $\boldsymbol{y}(\neq \boldsymbol{x})$ from $\boldsymbol{x}$ is given by*

$$\frac{1}{2} + \sum_{i=1}^{d_0} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} + \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}},$$

[3] In practice, if no such point exists, the algorithm should terminate. Here, our algorithm will run into an infinite loop, which fits well into our analytical framework.

where $d_0 \triangleq d(\boldsymbol{x}, \boldsymbol{y}) > 0$ *is the Hamming distance between* $\boldsymbol{x}$ *and* $\boldsymbol{y}$*. Furthermore, it is bounded above by*

$$\sum_{i=0}^{d_0} \binom{n}{i} + \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}}.$$

PROOF: Suppose the (1+1) EA-m keeps on generating points from $\boldsymbol{x}$ by standard mutation. The points so generated will form a list of $2^n - 1$ distinct elements, which excludes $\boldsymbol{x}$ itself. The position of $\boldsymbol{y}$ in the list is the remaining time to reach $\boldsymbol{y}$. Now we want to find the expected position of $\boldsymbol{y}$ in this random list. To determine it, note that

$$\text{position of } \boldsymbol{y} = 1 + \sum_{\boldsymbol{x}_j \neq \boldsymbol{x}, \boldsymbol{y}} I_j,$$

where

$$I_j = \begin{cases} 1, & \text{if } \boldsymbol{x}_j \text{ precedes } \boldsymbol{y} \\ 0, & \text{otherwise} \end{cases}$$

and so

$$\begin{aligned} E[\text{position of } \boldsymbol{y}] &= 1 + \sum_{\boldsymbol{x}_j \neq \boldsymbol{x}, \boldsymbol{y}} E[I_j] \\ &= 1 + \sum_{\boldsymbol{x}_j \neq \boldsymbol{x}, \boldsymbol{y}} \Pr\{\boldsymbol{x}_j \text{ precedes } \boldsymbol{y}\}. \end{aligned}$$

Let $d_j \triangleq d(\boldsymbol{x}, \boldsymbol{x}_j)$ and recall that $d_0 \triangleq d(\boldsymbol{x}, \boldsymbol{y})$. Then

$$\begin{aligned} \Pr\{\boldsymbol{x}_j \text{ precedes } \boldsymbol{y}\} &= \frac{\Pr[\boldsymbol{x}_j \text{ is generated from } \boldsymbol{x}]}{\Pr[\boldsymbol{x}_j \text{ is generated from } \boldsymbol{x}] + \Pr[\boldsymbol{y} \text{ is generated from } \boldsymbol{x}_0]} \\ &= \frac{\left(\frac{1}{n}\right)^{d_j}\left(1-\frac{1}{n}\right)^{n-d_j}}{\left(\frac{1}{n}\right)^{d_j}\left(1-\frac{1}{n}\right)^{n-d_j} + \left(\frac{1}{n}\right)^{d_0}\left(1-\frac{1}{n}\right)^{n-d_0}} \\ &= \frac{1}{1+(n-1)^{d_j-d_0}}. \end{aligned}$$

Hence,

$$\begin{aligned} E[\text{position of } \boldsymbol{y}] &= 1 + \sum_{i=1}^{d_0-1} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} + \left[\binom{n}{d_0} - 1\right]\frac{1}{2} + \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} \\ &= \frac{1}{2} + \sum_{i=1}^{d_0} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} + \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} \\ &\leq \sum_{i=0}^{d_0} \binom{n}{i} + \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}}. \end{aligned}$$

□

The RHS can be further simplified using the following lemma.

LEMMA 2: *The following two inequalities hold for any $d_0 > 0$:*

$$\sum_{i=0}^{d_0} \binom{n}{i} \leq 2^{rd_0}(1+2^{-r})^n,$$

for any $r > 0$, and

$$\sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} \leq \binom{n}{d_0+1}.$$

PROOF: For any positive number r, we have

$$2^{-rd_0} \sum_{i=0}^{d_0} \binom{n}{i} \leq \sum_{i=0}^{d_0} 2^{-ri} \binom{n}{i} \leq \sum_{i=0}^{n} 2^{-ri} \binom{n}{i} = (1+2^{-r})^n.$$

The first inequality then follows.

Next, we have

$$\begin{aligned}
&\sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} \\
&\leq \sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{(n-1)^{i-d_0}} \\
&= \frac{1}{n-1}\binom{n}{d_0+1} + \frac{1}{(n-1)^2}\binom{n}{d_0+2} + \cdots + \frac{1}{(n-1)^{n-d_0}}\binom{n}{n}.
\end{aligned}$$

It is straightforward to check that

$$\frac{1}{n-1}\binom{n}{k} \leq \binom{n}{k-1},$$

when $2 \leq k \leq n$. Hence,

$$\sum_{i=d_0+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-d_0}} \leq \frac{n-d_0}{n-1}\binom{n}{d_0+1} \leq \binom{n}{d_0+1}. \qquad \square$$

5 Unimodal Functions

In this section, we study the performance of various schemes on the class of unimodal functions, which has different definitions in the literature. Here we adopt the definition from Droste et al. (2002a).

DEFINITION 3: *A point $\boldsymbol{x}$ is a local maximum of a fitness function f if $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{B}_1(\boldsymbol{x})$. The function f is said to be unimodal if it has exactly one local maximum.*

For unimodal fitness functions, RLS_1, RLS_2, and RLS-m will not get trapped in suboptimal points. We do not explicitly consider RLS-m+, since at most one bit will be flipped during its execution, and hence it behaves exactly the same as RLS-m until an optimum has been found; in other words, $T_{\text{rm+}} = T_{\text{rm}}$ in distribution.

First, we consider the memory usage of RLS-m and (1+1) EA-m:

THEOREM 3: *The memory usage of RLS-m over unimodal functions is bounded above by n. The expected memory usage of (1+1) EA-m over unimodal functions is bounded above by en.*

PROOF: If an optimal point of a unimodal function has not been reached, RLS-m must be able to find another point within the 1-ball of the current point that can replace the current point. Once that point is found, the memory will be cleared. Hence, its memory usage is bounded above by n.

The expected memory usage of (1+1) EA-m, excluding the current point, is bounded above by the expected time for it to reach a better point within the 1-ball of the current point. Since the function is unimodal, such a point must exist and we denote it by $\boldsymbol{y}$. We make the pessimistic assumption that $\boldsymbol{y}$ is the only point that has a strictly larger fitness value than the current point. This means that only when $\boldsymbol{y}$ is generated can a fitness improvement be made. The expected time so obtained becomes an upper bound for the expected time that a fitness improvement is made, and is equal to

$$\frac{1}{\frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1}} = \frac{n}{\left(1-\frac{1}{n}\right)^{n-1}} \leq en. \qquad \square$$

Next, we consider the runtime of RLS-m over some pseudo-Boolean functions that are of particular interest. For completeness, the runtime of the (1+1) EA over our considered problems will also be stated. The runtime of (1+1) EA-m will be analyzed in the next section, when we consider multimodal fitness functions.

5.1 The OneMax Function

We first consider the function

$$\text{OneMax}(\boldsymbol{x}) \triangleq \sum_{i=1}^{n} x_i.$$

Its function value equals the number of ones in $\boldsymbol{x}$. Note that it is a stepwise unitation function. Hence, the choice of the replacement operator does not affect the result and we drop the subscript associated with RLS and (1+1) EA.

Without the use of memory, the worst-case runtimes of the (1+1) EA and RLS are unbounded. On the contrary, with the use of memory, we can avoid reevaluating some of the visited points. The following result gives an upper bound of T_{rm}.

THEOREM 4: *The runtime T_{rm} of RLS-m on* OneMax *is upper bounded by $n(n+1)/2$.*

PROOF: Suppose the current point has Hamming weight $k < n$. To find a point that can improve the fitness, at most $k+1$ points must be evaluated. The worst case occurs

when the initial point is the all-zero bit string. Hence,

$$T_{\mathrm{rm}} \leq \sum_{k=0}^{n-1}(k+1) = \frac{n(n+1)}{2}. \qquad \square$$

Although the problem space consists of 2^n points, the worst case of T_{rm} is just $O(n^2)$. Next we compare the expected runtimes of various schemes. The first two parts are not new and can be found in Garnier et al. (1999).

THEOREM 5: *Assume that the initial point is chosen uniformly over the entire state space.*

(a) The expected runtime of the (1+1) EA on ONEMAX *is*

$$E[T_s] \approx n(e \ln n + R) \qquad (n \to \infty),$$

up to terms of order $o(n)$, and R is a constant approximately equal to –1.90.

(b) The expected runtime of RLS on ONEMAX*, given that the Hamming distance between the initial point and the optimal point is k, is*

$$\tau_k \triangleq E[T_{\mathrm{r}} | d(\boldsymbol{x}^{(0)}, \boldsymbol{x}^*) = k] = \begin{cases} 0 & k = 0 \\ nH_k & k \geq 1 \end{cases}$$

where H_k is the kth harmonic number, defined as

$$H_k \triangleq \sum_{i=1}^{k} \frac{1}{i}.$$

Moreover, the expected runtime is

$$E[T_{\mathrm{r}}] \approx n(\ln n + \gamma - \ln 2) \quad (n \to \infty),$$

up to terms of order $o(n)$, and γ is Euler's constant, approximately equal to 0.577.

(c) The expected runtime of RLS-m on ONEMAX*, given that the Hamming distance between the initial point and the optimal point is k, is*

$$\tau_k^* \triangleq E[T_{\mathrm{rm}} | d(\boldsymbol{x}^{(0)}, \boldsymbol{x}^*) = k] = \begin{cases} 0 & k = 0 \\ (n+1)(H_{k+1} - 1) & k \geq 1 \end{cases}.$$

Moreover, the expected runtime is

$$E[T_{\mathrm{rm}}] \approx n(\ln n + \gamma - \ln 2 - 1) \quad (n \to \infty),$$

up to terms of order $o(n)$.

PROOF: (a) and (b). The proofs can be found in Garnier et al. (1999).

(c). If the Hamming distance between the initial point and the optimal point is k, then k steps are needed to reach the optimal point; in each step, the number of ones is increased by one. We first find the expected time to complete the first step.

Initially, there are k zeros and $n-k$ ones. RLS-m randomly chooses a bit to flip. As long as the bit chosen is a zero, the first step is completed. Since revisit is not allowed, the expected time for this to occur is the same as the expected time for the first occurrence of a zero in a uniformly random permutation of k zeros and $n-k$ ones. This in turn equals $M+1$, where M denotes the expected number of ones before the first zero and 1 denotes the position of the first zero. M can be found in the following way. Treat a "one" as a ball and a possibly empty consecutive sequence of ones as an urn. Put $n-k$ balls randomly into $k+1$ urns. M is equal to the expected number of balls in the first urn. By symmetry, the expected number of balls in all urns should be the same. Therefore, $M=(n-k)/(k+1)$. Hence, the expected completion time of the first step is $M+1=(n+1)/(k+1)$.

In total, there are k steps to reach the optimal point. The expected completion time of steps $2,3,\ldots,k$ are respectively $(n+1)/k,(n+1)/(k-1),\ldots,(n+1)/2$. Summing them up, we have for $k\geq 1$,

$$\tau_k^* = \sum_{i=1}^{k} \frac{n+1}{i+1} = (n+1)(H_{k+1}-1).$$

For $k=0$, we clearly have $\tau_k^*=0$.

The expected runtime can be written in terms of τ_k^* as follows:

$$\begin{aligned}
E[T_{\mathrm{rm}}] &= 2^{-n}\sum_{k=0}^{n}\binom{n}{k}\tau_k^* \\
&= 2^{-n}\sum_{k=1}^{n}\binom{n}{k}(n+1)(H_{k+1}-1) \\
&= (n+1)2^{-n}\sum_{k=1}^{n}\binom{n}{k}H_k - (n+1)2^{-n}(2^n\text{-}1) + 2^{-n}\sum_{k=1}^{n}\binom{n+1}{k+1} \\
&= \frac{n+1}{n}E[T_{\mathrm{r}}] - (n+1)(1-2^{-n}) + 2^{-n}\left(2^{n+1}-(n+1)-1\right) \\
&= \frac{n+1}{n}E[T_{\mathrm{r}}] - n + 1 - 2^{-n}.
\end{aligned}$$

Using the result in (b), we have

$$E[T_{\mathrm{rm}}] \approx n(\ln n + \gamma - \ln 2 - 1) \quad (n\to\infty),$$

up to terms of order $o(n)$. □

It can be seen that both RLS and RLS-m perform better than the (1+1) EA by a factor of e, for large n. The behavior of RLS-m is very similar to that of RLS. The only

difference is that reevaluating visited points will not occur in RLS-m. We can see that the largest relative improvement is obtained when $k = 1$. For that case, we have $\tau_1 = n$ and $\tau_1^* = (n + 1)/2$; the runtime is reduced by almost one half. In the other extreme, when $k = n$ and for large n, we have $\tau_n^*/\tau_n \to 1$. Therefore, the relative decrease in runtime by the memory-assisted version diminishes to zero. The reason is that when k is large, there are more 0s; it is more probable that a 0 is chosen for flipping, which leads to improvement, and the chance of the occurrence of the same 1 chosen more than once is infrequent. Hence, the advantage of avoiding revisit is not great. When the initial point is uniformly distributed among the entire space, the percentage decrease in expected runtime is approaching zero when n is approaching infinity.

5.2 Linear Functions

The ONEMAX function belongs to the class of linear functions which is defined below.

DEFINITION 4: *A function* $f : \{0, 1\}^n \to \Re$ *is said to be linear if it can be written as*

$$f(x) = \sum_{i=1}^{n} w_i x_i,$$

for some $w_i \in \Re_+$.

There is no loss of generality in assuming that all weights are nonnegative, for otherwise, if $w_i < 0$, we may simply redefine x_i by $1 - x_i$. Note that the all-one vector is a global maximum. Furthermore, if all weights are nonzero, then it is the only global maximum.

We analyze the performance of RLS and RLS-m over linear functions of nonzero weights. Note that functions of this class are stepwise functions, and RLS has the same runtime result for both replacement operators.

THEOREM 6: *The runtime of RLS (or RLS-m) over any linear function of nonzero weight has the same probability distribution as that over* ONEMAX.

PROOF: Given any linear function f of nonzero weights, the following identity holds:

$$\Lambda(f(\boldsymbol{x}), f(\boldsymbol{x}_0)) \equiv \Lambda(\text{ONEMAX}(\boldsymbol{x}), \text{ONEMAX}(\boldsymbol{x}_0)),$$

for any $\boldsymbol{x} \in \mathcal{B}_1(\boldsymbol{x}_0)$. Hence, if $\boldsymbol{x}_0$ is the current point and $\boldsymbol{x}$ the newly generated point, RLS (or RLS-m) will replace $\boldsymbol{x}_0$ by $\boldsymbol{x}$ when optimizing f if and only if it will do so when optimizing ONEMAX. □

Therefore, results for ONEMAX stated in Theorem 5(b) and (c) can be applied to the entire class of linear functions with nonzero weights. We remark that the runtime of (1+1) EA over linear functions with nonzero weights is $\Theta(n \log n)$ (Droste et al., 2002a).

5.3 The LeadingOnes Function

The percentage decrease in expected runtime by RLS-m can be strictly positive when compared with RLS. To illustrate this fact, we consider the function

$$\textsc{LeadingOnes}(\boldsymbol{x}) \triangleq \sum_{i=1}^{n} \prod_{j=1}^{i} x_j,$$

whose function value equals the number of leading ones in $\boldsymbol{x}$. Before going into the issue of runtime reduction, we derive an upper bound for T_{rm} first.

THEOREM 7: *The runtime T_{rm} of RLS-m on* LeadingOnes *is upper bounded by n^2.*

PROOF: To increase the fitness value, the first leading bit has to be chosen for flipping. In the worst case, n iterations are needed to choose the leading bit correctly. The worst case occurs when the initial point is the all-zero bit string. Hence, totally n steps are needed, and so $T_{\mathrm{rm}} \leq n^2$. □

Next we consider the expected runtime of the three algorithms. Although LeadingOnes is neither a stepwise nor a unitation function, the following result shows that for both (1+1) EA and RLS, their expected runtimes are the same for both replacement operators.

THEOREM 8: *Assume that the initial point is chosen uniformly over the entire state space.*

(a) The expected runtime of the (1+1) EA on LeadingOnes *is*

$$\begin{aligned} E[T_s] &= \frac{n(n-1)}{2}\left[\left(1-\frac{1}{n}\right)^{-n} - 1\right] \\ &= \frac{(e-1)n^2}{2} \quad (n \to \infty). \end{aligned}$$

(b) The expected runtime of RLS on LeadingOnes *is*

$$E[T_{\mathrm{r}}] = \frac{n^2}{2}.$$

(c) The expected runtime of RLS-m on LeadingOnes *is*

$$E[T_{\mathrm{rm}}] = \frac{n(n+1)}{4}.$$

PROOF: (a) Suppose that there are k leading ones after a certain number of iterations. We call it stage k. When standard mutation is used, the probability that the $(k+1)$th bit is flipped while the first k bits are not is equal to $(1-1/n)^k(1/n)$. When the first $(k+1)$ bits are all ones, the probability that the $(k+2)$th bit is also one is one half, meaning that the probability that stage $k+1$ can be bypassed is one half. The probability is independent of all previous events that occur in stage k and before, because all such

events leave the value of the $(k+1)$th bit uniformly random, with equal probability of being 0 or 1. Hence, the expected runtime is

$$E[T_s] = \sum_{k=0}^{n-1} \frac{1}{2}\left(1-\frac{1}{n}\right)^{-k} n + \frac{1}{2}(0)$$
$$= \frac{n(n-1)}{2}\left[\left(1-\frac{1}{n}\right)^{-n} - 1\right].$$

The above argument applies to both replacement operators.

(b) We can apply the same argument as in (a). The only difference is that the probability of success in stage k should be replaced by $1/n$, because of the change of mutation operator. Hence,

$$E[T_r] = \sum_{k=0}^{n-1} \frac{n}{2} = \frac{n^2}{2}.$$

(c) When RLS-m is used, a revisit of previous points will not occur. Hence, given an initial point, the expected time for flipping the correct bit is the same as the expected position of a particular bit in a random permutation of n bits, which is equal to $(n+1)/2$. Suppose the initial bit string consists of K zeros. The conditional expected runtime is

$$E[T_{rm}|K=k] = \begin{cases} k\left(\frac{n+1}{2}\right) & 1 \le k \le n; \\ 0 & k = 0. \end{cases}$$

Let p_k be the probability that $K = k$.

$$E[T_{rm}] = \sum_{k=1}^{n} p_k \left[k\left(\frac{n+1}{2}\right)\right]$$
$$= \frac{n+1}{2}E[K]$$
$$= \frac{n(n+1)}{4}.$$

□

It can be seen that for large n, RLS-m reduces the expected runtime of RLS by 50%. In the next section, we will show that this is the best percentage reduction that we can hope for when applying RLS-m to unimodal functions.

5.4 General Case

In this section, we compare the relative performance of RLS_2 and RLS-m. For RLS, we consider the type-2 replacement operator only, because it is adopted in RLS-m. We have the following general result for unimodal fitness functions.

THEOREM 9: *Given any unimodal fitness function f, the expected runtime for RLS$_2$ and RLS-m on f satisfy*

$$\frac{1}{2} < \frac{E[T_{\mathrm{rm}}]}{E[T_{\mathrm{r2}}]} \leq 1.$$

PROOF: The upper bound is obvious. We are going to prove the lower bound. Our proof consists of two basic arguments. First, the probabilities of occurrence of a particular trajectory under RLS$_2$ and RLS-m are the same. Second, the relative time difference for RLS$_2$ and RLS-m to proceed to the next better neighbor under a given trajectory can be bounded.

The first argument is true because the mechanism of generating new points for fitness evaluation under the two algorithms is the same.[4] We formalize this observation as follows. We say that an iterative algorithm proceeds to the next stage whenever there is a strict increase in the fitness value. Initially, let an algorithm be in stage 1. Let $\mathcal{P}_{\mathrm{r}}(t)$ and $\mathcal{P}_{\mathrm{rm}}(t)$ be the finite sequence of points generated respectively by RLS$_2$ and RLS-m up to iteration t.[5] Recall that the fitness values of points in each sequence are nondecreasing. Moreover, let $\mathcal{Q}_{\mathrm{r}}(t)$ and $\mathcal{Q}_{\mathrm{rm}}(t)$ be the longest finite subsequences[6] of $\mathcal{P}_{\mathrm{r}}(t)$ and $\mathcal{P}_{\mathrm{rm}}(t)$ that consist of distinct elements. By definition, the fitness values of points in $\mathcal{Q}_{\mathrm{r}}(t)$ and $\mathcal{Q}_{\mathrm{rm}}(t)$ are strictly increasing. For a particular instance, if the length of $\mathcal{Q}_{\mathrm{r}}(t)$ is K, then the algorithm has gone through K stages. If in addition, the optimal point is in $\mathcal{Q}_{\mathrm{r}}(t)$, then the optimal point, $\boldsymbol{x}^*$, is reached after entering stage K and $\boldsymbol{x}^*$ must be its last element. We drop parameter t and denote that sequence by $\mathcal{Q}_{\mathrm{r}}$, meaning that t is sufficiently large that the optimal point has been reached. The same applies to $\mathcal{Q}_{\mathrm{rm}}(t)$, and we define $\mathcal{Q}_{\mathrm{rm}}$ similarly. Our first argument merely says that the distributions of $\mathcal{Q}_{\mathrm{r}}$ and $\mathcal{Q}_{\mathrm{rm}}$ are the same. We denote the common probability mass function by $p(\mathcal{Q})$.

Given that $\mathcal{Q}_{\mathrm{r}} = \mathcal{Q} \triangleq \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{|\mathcal{Q}|}\}$, let $\tau_{\mathrm{r}}(k|\mathcal{Q})$ be the random variable representing the number of iterations required for RLS to change from stage k to stage $k+1$, for $k = 1, 2, \ldots, |\mathcal{Q}| - 1$. We define $\tau_{\mathrm{rm}}(k|\mathcal{Q})$ in a similar way. Our second argument is that we can bound the ratio of these two quantities.

We consider RLS$_2$ first. For $k = 1, 2, \ldots, |\mathcal{Q}| - 1$, let l_k be the number of points in $\mathcal{B}_1(\boldsymbol{x}_k)$ that have strictly larger fitness values than $\boldsymbol{x}_k$ does. Note that $l_k \geq 1$ for any nonoptimal $\boldsymbol{x}_k$, since f is unimodal. Among these l_k points, one of them is $\boldsymbol{x}_{k+1}$. Given that $\boldsymbol{x}_{k+1}$ takes the algorithm to stage $k+1$, we know that the other $l_k - 1$ points in $\mathcal{B}_1(\boldsymbol{x}_k)$ had not occurred in stage k. The number of possible points that could have been generated in stage k, including $\boldsymbol{x}_{k+1}$ and excluding $\boldsymbol{x}_k$, is $n - l_k + 1$. Since the occurrence of each of them is equiprobable, the probability (conditioned on $\mathcal{Q}$) that $\boldsymbol{x}_{k+1}$ is generated is $1/(n - l_k + 1)$. As a result,

$$E[\tau_{\mathrm{r}}(k|\mathcal{Q})] = n - l_k + 1.$$

A similar argument applies to RLS-m. Given $\mathcal{Q}$, the number of possible points that could have been generated in stage k is $n - l_k + 1$. Since visited points will not

[4]The only difference is that RLS-m discards revisited points.

[5]A precise definition can be found in Section 2.

[6]Recall that a subsequence of $\mathcal{S}$ is a sequence that can be derived from $\mathcal{S}$ by deleting some of its elements without changing the order of the remaining elements.

be generated again, the expected time for the occurrence of $\boldsymbol{x}_{k+1}$ is the same as the expected position of a particular bit in a random permutation in a string with $n - l_k + 1$ bits. Hence, we have

$$E[\tau_{\mathrm{rm}}(k|\mathcal{Q})] = \frac{n - l_k + 2}{2},$$

for $1 \leq k \leq |\mathcal{Q}| - 1$.

Combining the above two equations, $E[\tau_{\mathrm{rm}}(k|\mathcal{Q})] > E[\tau_{\mathrm{r}}(k|\mathcal{Q})]/2$ for all k. Hence,

$$E[T_{\mathrm{rm}}] = \sum_{\mathcal{Q}} p(\mathcal{Q}) \sum_{k=1}^{|\mathcal{Q}|-1} E[\tau_{\mathrm{rm}}(k|\mathcal{Q})] > \frac{1}{2} \sum_{\mathcal{Q}} p(\mathcal{Q}) \sum_{k=1}^{|\mathcal{Q}|-1} E[\tau_{\mathrm{r}}(k|\mathcal{Q})] = \frac{1}{2} E[T_{\mathrm{r}}]. \qquad \square$$

Recall that when n is large, $E[T_{\mathrm{rm}}]/E[T_{\mathrm{r}}]$ tends to $1/2$ and 1 for LeadingOnes and OneMax, respectively. These examples show that both the lower bound and the upper bound in the above theorem are asymptotically tight.

6 Multimodal Functions

In this section, we consider three functions that belong to the class of multimodal functions, which is defined below.

Definition 5: *A function f is said to be multimodal if it has two or more local maxima.*

For multimodal fitness functions, both RLS and RLS-m may be trapped in local maxima since at most one bit is flipped in each iteration. In that case, their running time will grow to infinity. Hence, in this section, we only consider the performance of the (1+1) EA, the (1+1) EA-m, and RLS-m+. For comparison, we may also consider a simple random search, which generates points uniformly at random in the problem space. Given any fitness function that has a unique global maximum, the random search always generates the optimal point with probability 2^{-n}. Without counting the initial point, the expected runtime of such a naive strategy is $2^n - 1$. We will see that the memory-assisted algorithms perform better than both the (1+1) EA and random search.

While the average memory requirement for RLS-m and (1+1) EA-m over unimodal functions are $O(n)$ and $O(n^2)$, respectively, their memory requirement over multimodal functions can be very large. Clearly, their expected memory requirement over a given function is bounded by their expected runtime over that function, since the runtime is equal to the number of distinct points whose fitness values a memory-assisted algorithm has evaluated. Hence, we have the following result.

Theorem 10: *The expected memory requirement for RLS-m and (1+1) EA-m are $O(T_{\mathrm{rm}})$ and $O(T_{\mathrm{sm}})$, respectively.*

Therefore, for the following examples, we will only consider expected runtimes.

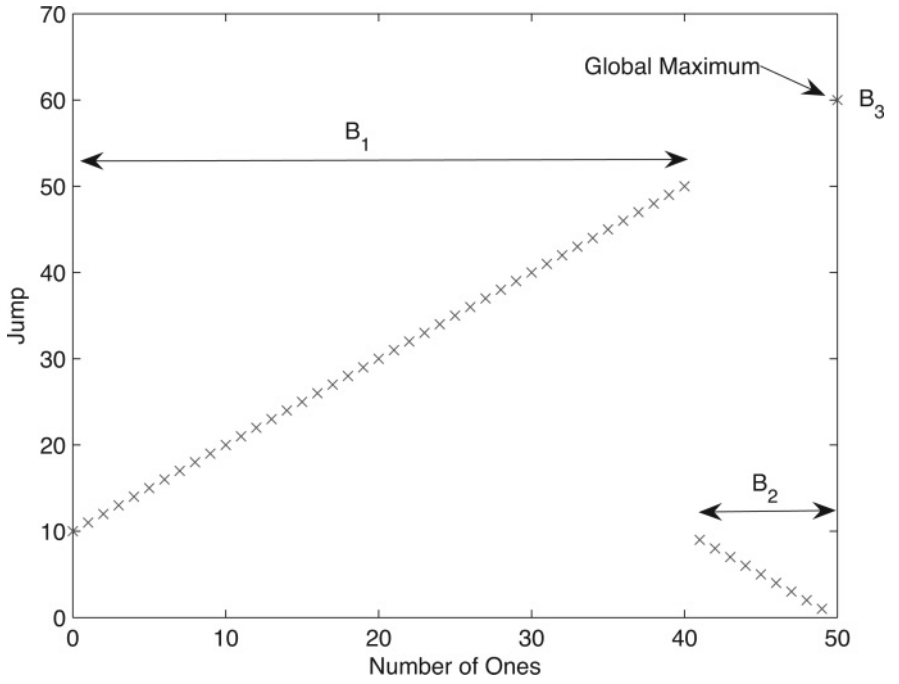

Figure 1: The function $\textsc{Jump}_{m,n}$ for $n = 50$ and $m = 10$.

6.1 The Jump Function

For $m = 1, 2, \ldots, n$, define the following function (Droste et al., 2002a).

$$\textsc{Jump}_{m,n}(\boldsymbol{x}) \triangleq \begin{cases} m + \sum_{i=1}^{n} x_i & \text{if } \sum_{i=1}^{n} x_i \leq n - m \\ & \text{or } \sum_{i=1}^{n} x_i = n, \\ n - \sum_{i=1}^{n} x_i & \text{otherwise.} \end{cases}$$

The global maximum of this function occurs at 1. In total, there are $\binom{n}{m}$ suboptimal local maxima, which are the points that have Hamming weight $n - m$. Note that the Hamming distance between a local maximum and the global maximum is m. For easy visualization, we plot the function in Figure 1 for $n = 50$ and $m = 10$. Note that this is an injective unitation function. Thus we omit the subscript associated with the replacement operators.

We are interested in studying the order of growth of the expected runtime on this function when m and n go to infinity at the same rate. In other words, the ratio $\alpha \triangleq m/n$ is assumed to be a constant independent of m and n.

Theorem 11: *Let $0 < \alpha \leq 1$, and $m \triangleq \alpha n$ be a positive integer.*

(a) *The expected runtime of the (1+1) EA on* $\textsc{Jump}_{\alpha n,n}$ *is* $\Theta(n^{\alpha n})$.

(b) *The expected runtime of the (1+1) EA-m on* $\textsc{Jump}_{\alpha n,n}$ *is* $O(2^{nh_{\mathrm{b}}(\min\{\alpha,1/2\})})$, *where* $h_{\mathrm{b}} : [0,1] \to \Re$ *is the binary entropy function defined as*

$$h_{\mathrm{b}}(x) \triangleq -x\log_2 x - (1-x)\log_2(1-x).$$

(c) *The expected runtime of RLS-m+ on* $\textsc{Jump}_{\alpha n,n}$ *is also* $O(2^{nh_{\mathrm{b}}(\min\{\alpha,1/2\})})$.

PROOF: (a) By Droste et al. (2002a), Theorem 25, we have $T_s = \Theta(n^{\alpha n} + n\log n)$, which in turn equals $\Theta(n^{\alpha n})$.

(b) Following Droste et al. (2002a), we partition the search space into three disjoint sets $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ with

$$\mathcal{B}_1 \triangleq \left\{ \boldsymbol{x} \in \{0,1\}^n \middle| \sum_{i=1}^{n} x_i \le n-m \right\},$$

$$\mathcal{B}_2 \triangleq \left\{ \boldsymbol{x} \in \{0,1\}^n \middle| n-m < \sum_{i=1}^{n} x_i < n \right\},$$

$$\mathcal{B}_3 \triangleq \{\mathbf{1}\}.$$

Suppose that the initial point belongs to $\mathcal{B}_1$. The function is similar to ONEMAX, and the (1+1) EA will reach a point with exactly m zeros after $O(n\log n)$ steps on average. The situation is similar if the initial point belongs to $\mathcal{B}_2$. In that case, the zero bits will be flipped until it reaches a point with exactly m zeros or the global maximum. Again the expected time is $O(n\log n)$. Therefore, on average, after $O(n\log n)$ steps, either the global maximum or a local maximum is reached. Under the (1+1) EA-m, visited points will not be evaluated again. Therefore, the number of additional points that need to be evaluated until the global maximum is reached cannot exceed 2^n. This proves the statement for $\alpha \ge 1/2$.

Next we consider the case where $\alpha < 1/2$. We assume that the algorithm has already reached one of the local maxima. The global maximum will be reached when 1 is generated from the local maximum. Note that the Hamming distance between them is m. By Lemma 1, the expected remaining runtime is bounded above by

$$\sum_{i=0}^{m} \binom{n}{i} + \sum_{i=m+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-m}}. \tag{3}$$

By Lemma 2, we have

$$\sum_{i=0}^{m} \binom{n}{i} \le (2^{r\alpha} + 2^{r(\alpha-1)})^n.$$

Substituting $r = \log_2\big((1-\alpha)/\alpha\big)$, we get

$$\sum_{i=0}^{m} \binom{n}{i} \le 2^{nh_{\mathrm{b}}(\alpha)}\left[2^{\log_2(1-\alpha)} + 2^{\log_2\alpha}\right]^n = 2^{nh_{\mathrm{b}}(\alpha)}. \tag{4}$$

By Lemma 2 again, the second term in Equation (3) can be bounded as follows:

$$\sum_{i=m+1}^{n} \binom{n}{i} \frac{1}{1+(n-1)^{i-m}} \leq \binom{n}{m+1} = \frac{n-m}{m+1}\binom{n}{m} \leq \frac{1-\alpha}{\alpha}\binom{n}{m} \leq \frac{1-\alpha}{\alpha} 2^{nh_{\mathrm{b}}(\alpha)},$$

where the last inequality follows from Equation (4).

(c) The proof is similar to that for (b). On average, after $O(n \log n)$ steps, either the global maximum or a local maximum is reached. Once a point with exactly m zeros is reached, the upper bound of the number of additional points that need to be evaluated until the global maximum is reached is

$$\sum_{i=1}^{m} \binom{n}{i},$$

which is obviously less than 2^n. This proves the statement for the case where $\alpha \geq 1/2$. For the case where $\alpha < 1/2$, we get the upper bound $2^{nh_{\mathrm{b}}(\alpha)}$ from Equation (4). □

The above example shows that both the (1+1) EA-m and RLS-m can reduce the order of growth of the expected runtime, when compared with the (1+1) EA. Besides, they perform better than random search when α is less than one half.

6.2 The Peak Function

Next we consider the following function:

$$\textsc{Peak}(\boldsymbol{x}) \triangleq \prod_{i=1}^{n} x_i.$$

This function has only one peak at 1, while all other bit strings form a large plateau. This problem is also called a needle in the haystack. The function is multimodal, since all points except those in $\mathcal{B}_1(1) \setminus \{1\}$ are local maxima. Since there is a large plateau, this function was used in Droste et al. (2002a) to illustrate that the order of growth of the expected runtime can be reduced if (1+1) EA_2 is changed into (1+1) EA_1. With that change, the expected runtime is upper bounded by $2^n(1 + o(1))$. Here we show that avoiding the evaluation of visited points can serve the same purpose.

Theorem 12: *Assume that the initial point is chosen uniformly over the entire state space.*

(a) The expected runtime of (1+1) EA_1 on Peak *is upper bounded by* $2^n(1 + o(1))$.

(b) The expected runtime of (1+1) EA_2 on Peak *is lower bounded by* $e^{n \ln(n/2)}$.

(c) The expected runtimes of (1+1) EA-m and RLS-m+ on Peak *are both* $2^{n-1} - \frac{1}{2}$.

Proof: The proofs of (a) and (b) can be found in Droste et al. (2002a).

The proof of (c) is the same for both (1+1) EA-m and RLS-m+. First of all, let $\boldsymbol{x}^{(0)}$ be the initial point. The problem is equivalent to another problem where $\mathbf{0}$ is the initial point, and the peak changes from $\mathbf{1}$ to $\boldsymbol{x}^*$, where $\boldsymbol{x}^*$ is equal to the exclusive-OR between

$\boldsymbol{x}^{(0)}$ and $\mathbf{1}$. Since in the original problem, the initial point is chosen uniformly, in the equivalent problem, the optimal point $\boldsymbol{x}^*$ can be regarded as uniformly distributed over all points. Since for this problem, a visited point will never be evaluated again, both the (1+1) EA-m and RLS-m+ will generate distinct points until $\boldsymbol{x}^*$ is reached. No matter how the sequence is generated, the probability of reaching $\boldsymbol{x}^*$ in the kth step is equal to 2^{-n} for $k = 1, 2, \ldots, (2^n - 1)$. Hence, the expected runtimes for both algorithms are the same and equal to

$$2^{-n}\left[1 + 2 + \cdots + (2^n - 2) + (2^n - 1)\right] = 2^{n-1} - \frac{1}{2}. \qquad \square$$

For the PEAK function, the expected runtime of RLS-m has the same order of growth as random search. On the other hand, (1+1) EA-m and RLS-m do have a 50% reduction when compared with random search. In fact, they are both optimal, since their expected runtimes are the same as the black box complexity of the function class defined by PEAK (Wegener, 2005, p. 121).[7]

6.3 The ONEMAX-ZEROSPIKE Function

Consider this function:

$$\text{ONEMAX-ZEROSPIKE}(\boldsymbol{x}) \triangleq \begin{cases} \text{ONEMAX}(x) & x \neq 0 \\ n - 1/2 & x = 0. \end{cases}$$

This function is the same as ONEMAX except at $x = 0$. It does not give any misleading hints regarding the position of its global maximum, which occurs at 1. However, there is a spike at 0, since ONEMAX-ZEROSPIKE(0) > ONEMAX-ZEROSPIKE($\boldsymbol{x}$) for all $x \neq 1$. Moreover, it is an injective unitation function.

The next result shows that RLS-m+ can reduce the order of growth of the expected runtime from exponential to polynomial.

THEOREM 13: *Assume that the initial point is chosen uniformly over the entire state space.*

(a) The expected runtime of the (1+1) EA on ONEMAX-ZEROSPIKE *is* $\Omega((n/2)^n)$.

(b) The expected runtime of RLS-m+ on ONEMAX-ZEROSPIKE *is* $O(n \log n)$.

PROOF: (a) The probability that the initial point is $\mathbf{0}$ is 2^{-n}. Starting from 0, the expected time to move to 1 is n^n. Hence, $E[T_s] \geq (n/2)^n$.

(b) There are only two possibilities that the RLS-m+ can reach $\mathbf{0}$: the initial point being 0, or the initial point having a single 1, which was flipped in the first generation. The probabilities of these two events are respectively 2^{-n} and $(n/2^n)(1/n) = 2^{-n}$. Should these two events occur, the time to reach the global maximum is upper bounded by 2^n, which is just the number of points in the entire search space. Otherwise, the function is the same as ONEMAX and the expected runtime is upper bounded by $Cn \log n$, for some

[7]Note that in our definition of runtime, the evaluation of the initial point is not counted, which is different from that in Wegener (2005).

positive constant C. Hence,

$$\begin{aligned} E[T_{\mathrm{rm}+}] &\leq (2^{-n} + 2^{-n})2^n + (1 - 2^{-n} - 2^{-n})Cn \log n \\ &= 2 + (1 - 2^{-n+1})Cn \log n \\ &= O(n \log n). \end{aligned}$$

□

We conjecture that the expected runtimes of (1+1) EA-m and RLS-m+ are of the same order. However, a proof has not yet been found.

7 A Unified Framework

The methodology and analytical tools used in previous sections can be applied to other (1+1) EAs. In this section, we propose a unified framework. To this end, we first redefine the notion of local maximum. Given a (1+1) EA with genetic operator ϕ, we define the neighborhood of a point x, $\mathcal{N}_\phi(x)$ as the set of points that can be generated from x using ϕ.

DEFINITION 6: *A point x is a local maximum of a fitness function f with respect to ϕ if $f(x) \geq f(y)$ for all $y \in \mathcal{N}_\phi(x)$. The function f is said to be unimodal with respect to ϕ if it has exactly one local maximum.*

Note that our previous definition on local maximum is a special case of the above, with ϕ representing the one-bit flip. For the (1+1) EA, the corresponding ϕ is the standard mutation. Under this operator, all the points in the domain can be reached. Hence, given any fitness function, a local maximum must be a global maximum. If there is a unique global maximum, then the fitness function is unimodal. Hence, the JUMP, PEAK and ONEMAX-ZEROSPIKE functions are all unimodal with respect to the standard mutation. In this section, we will analyze the relative performance of memory-assisted algorithms over unimodal functions with respect to their genetic operators.

Denote the (1+1) EA with genetic operator ϕ by Φ_0, and its memory-assisted version by Φ_m. Denote their runtimes by T_0 and T_{m}, respectively. We further assume that Φ_0 has a spatially invariant neighborhood, as defined below.

DEFINITION 7: *A (1+1) EA with genetic operator ϕ is said to have a spatially invariant neighborhood if given any point x, $\mathcal{N}_\phi(x)$ always consists of the same number of points, and the probability mass function of those points are always the same up to relabeling.*

Note that both the (1+1) EA and RLS have spatially invariant neighborhoods.

Let N be the size of the neighborhood. We assume that the probability masses are sorted in descending order, that is, $p_1 \geq p_2 \geq \cdots \geq p_N$. We have the following general result.

THEOREM 14: *Given a (1+1) EA$_2$, Φ_0, with genetic operator ϕ and a spatially invariant neighborhood, and its memory-assisted version Φ_{m}, their respective expected runtimes, T_0 and*

T_{m}, over any unimodal function (with respect to ϕ) satisfy

$$p_N + \sum_{i=1}^{N-1} \frac{p_i p_N}{p_i + p_N} \leq \frac{E[T_{\mathrm{m}}]}{E[T_0]} \leq 1,$$

where $p_1, p_2, \ldots, p_N$ are the probability masses of the neighborhood, sorted in descending order.

PROOF: The upper bound is trivial. Now we prove the lower bound. The idea is the same as that in Theorem 9. We first note that the probabilities of occurrence of a particular trajectory under Φ_0 and Φ_{m} are the same. Therefore, we can focus on the number of iterations needed for Φ_0 and Φ_{m} to change from stage k to stage $k+1$.

Let the current point at stage k be x. Suppose that there are K points in $\mathcal{N}_\phi(x)$, which have strictly larger fitness than x. Denote these K points by $y_1, y_2, \ldots, y_K$, and their probability masses by $p_{i_1}, p_{i_2}, \ldots, p_{i_K}$. Define $\mathcal{K}$ be the set of these K points and $P \triangleq \sum_{r=1}^{K} p_{i_r}$. Denote the expected time to evolve from stage k to stage $k+1$, under Φ_0 and Φ_{m} by τ_{0k} and $\tau_{\mathrm{m}k}$, respectively. Then

$$\tau_{0k} = \frac{1}{P}$$

and

$$\tau_{\mathrm{m}k} = 1 + \sum_{j \in \mathcal{N}_\phi(\boldsymbol{x}) \setminus \mathcal{K}} E[I_j],$$

where I_j indicates whether the jth element in the neighborhood precedes all points in $\mathcal{K}$. Hence,

$$\tau_{\mathrm{m}k} = 1 + \sum_{j \in \mathcal{N}_\phi(\boldsymbol{x}) \setminus \mathcal{K}} \frac{p_j}{p_j + P}.$$

The ratio is then given by

$$\frac{\tau_{\mathrm{m}k}}{\tau_{0k}} = P + \sum_{j \in \mathcal{N}_\phi(\boldsymbol{x}) \setminus \mathcal{K}} \frac{p_j P}{p_j + P}.$$

We are going to show that the above ratio is the smallest when $K = 1$. Suppose on the contrary that $K > 1$. Since the expression $ax/(a+x)$ is increasing with x, it can be shown that the expression on the right-hand side is greater than

$$P - p_{\mathrm{m}} + \sum_{j \in \mathcal{N}_\phi(\boldsymbol{x}) \setminus \mathcal{K}} \frac{p_j (P - p_{\mathrm{m}})}{p_j + (P - p_{\mathrm{m}})} + \frac{p_{\mathrm{m}} (P - p_{\mathrm{m}})}{p_{\mathrm{m}} + (P - p_{\mathrm{m}})}.$$

This means that the ratio becomes smaller if one element is removed from $\mathcal{K}$. By repeating this argument, the ratio between $\tau_{\mathrm{m}k}$ and τ_{0k} is minimized when $K = 1$.

When there is only one element in $\mathcal{K}$, it is simple to show that the ratio is minimized by $P = p_N$, and thus is at least

$$p_N + \sum_{i=1}^{N-1} \frac{p_i p_N}{p_i + p_N}.$$

As the above argument applies to every stage where there is a strict fitness improvement, the ratio between the two expected times is also bounded below by the above expression. □

Theorem 9 then comes immediately from the following corollary.

COROLLARY 1: *If the probability distribution of the neighborhood is uniform, that is,* $p_1 = p_2 = \cdots = p_N$, *then*

$$\frac{1}{2} < \frac{1}{2}\left(1 + \frac{1}{N}\right) \leq \frac{E[T_{\mathrm{m}}]}{E[T_0]} \leq 1.$$

To illustrate the use of Theorem 14, we define a new (1+1) EA whose genetic operator randomly flips one or two bits. We let $\mathcal{R}_k(\boldsymbol{x})$ be the k ring of x defined by $\{y : d(x, y) = k\}$. With probability p, a point will be generated uniformly at random from $\mathcal{R}_1(x)$. With probability $q \triangleq 1 - p$, a point will be generated uniformly at random from $\mathcal{R}_2(x)$. We call this algorithm RLS with neighborhood $\mathcal{B}_2$, and denote it by $\mathrm{RLS}_2(\mathcal{B}_2)$; we have assumed that the type-2 replacement operator is used. Its memory-assisted version is denoted by RLS-m($\mathcal{B}_2$). Note that in general, p can be a function of the length of the input bit string, that is, $p \equiv p(n)$. Here we assume that $p(n) \geq 1/2$ for all n.

We now analyze their runtime performance over the following function:

$$\textsc{LeadingCouples}(\boldsymbol{x}) \triangleq \sum_{i=1}^{n/2} \prod_{j:odd,\, j=1}^{2i} x_j \bar{x}_{j+1},$$

for n being a positive, even integer. In other words, its function value equals the number of leading 10 pairs in x. There is one single global maximum at $(10)^{n/2}$, where the notation b^n represents the repetition of b by n times.

We first apply Theorem 14 to find a lower bound of the relative performance between $\mathrm{RLS}_2(\mathcal{B}_2)$ and RLS-m($\mathcal{B}_2$). We denote their runtimes by T_{r2} and T_{rm}, respectively.

COROLLARY 2: *The expected runtimes of* $RLS_2(\mathcal{B}_2)$ *and RLS-m*($\mathcal{B}_2$) *on* LEADINGCOUPLES *satisfy*

$$\frac{E[T_{\mathrm{rm}}]}{E[T_{\mathrm{r2}}]} = \Omega(q),$$

if $p(n) \geq 1/2$ *for all* n.

PROOF: For this particular case, we have $N = n + \binom{n}{2}$, and

$$p_i = \begin{cases} p/n & 1 \le i \le n, \\ q/\binom{n}{2} & \text{otherwise.} \end{cases}$$

Applying Theorem 14, we have

$$\begin{aligned} \frac{E[T_{\mathrm{rm}}]}{E[T_{\mathrm{r2}}]} &\ge \frac{q}{\binom{n}{2}} + n\frac{(p/n)\left(q/\binom{n}{2}\right)}{p/n + q/\binom{n}{2}} + \left[\binom{n}{2} - 1\right]\frac{q}{2\binom{n}{2}} \\ &= \frac{q}{n(n-1)} + \frac{2pq}{(n-1)p + 2q} + \frac{q}{2}. \end{aligned}$$

The statement then follows because the last term in the above expression dominates. □

Corollary 2 implies that it is possible for the memory-assisted version to reduce the order of growth of the expected runtime on LEADINGCOUPLES, provided that q is small enough, say $q = 1/n$. Now we perform an analysis to see if it is true indeed.

THEOREM 17: *Assume that the initial point is chosen uniformly over the entire state space, and* $p(n) \ge 1/2$ *for all* n.

(a) The expected runtime of $RLS_2(\mathcal{B}_2)$ *on* LEADINGCOUPLES *is*

$$E[T_{\mathrm{r2}}] = \frac{n^2}{4}\left[\frac{1}{p} + \frac{n-1}{4q}\right] = \Theta\left(\frac{n^3}{q}\right).$$

(b) The expected runtime of RLS-$m(\mathcal{B}_2)$ *on* LEADINGCOUPLES *is*

$$E[T_{\mathrm{rm}}] = \frac{n}{32}\left[n^2 + 3n - 6 + \frac{4n(n-1)(1+q)}{n-1-(n-3)q}\right] = \Theta(n^3).$$

PROOF: (a) The algorithm needs to pass through $n/2$ stages. In order to pass through a stage, there are three possibilities. First, a stage is bypassed without flipping any bit, which occurs with probability $1/4$. Second, a stage is passed through by flipping one and only one bit, which occurs with probability $1/2$. Third, a stage is passed by flipping two bits, which occurs with probability $1/4$. The expected time to pass through each stage is the same and is equal to

$$\frac{1}{4}(0) + \frac{1}{2}\left(\frac{n}{p}\right) + \frac{1}{4}\left(\frac{\binom{n}{2}}{q}\right) = \frac{n}{2p} + \frac{n(n-1)}{8q}.$$

Hence,

$$E[T_{\mathrm{r2}}] = \frac{n^2}{4}\left[\frac{1}{p} + \frac{n-1}{4q}\right].$$

Since $p \geq 1/2$, the second term dominates. Hence, $E[T_{\mathrm{r2}}] = \Theta(n^3/q)$.

(b) The algorithm needs to pass through $n/2$ stages and there are three possibilities as described in (a). If the first case occurs, the expected time is clearly zero. Now we consider the second case. Totally, there are $n + \binom{n}{2}$ points in the neighborhood of the current point, x. Let $x' \in \mathcal{R}_1(x)$ be the desired point that can take the algorithm to the next stage. The probability that a point in $\mathcal{R}_1(x)\backslash\{x'\}$ is generated before x' is $1/2$, and the probability that a point in $\mathcal{R}_2(x)$ is generated before x' is

$$\frac{q/\binom{n}{2}}{q/\binom{n}{2} + p/n}.$$

Therefore, if the second case occurs, the expected time is

$$\tau_1 = \frac{n-1}{2} + \binom{n}{2}\frac{q/\binom{n}{2}}{q/\binom{n}{2} + p/n} = \Theta(n^2).$$

Similarly, if the third case occurs, the expected time is

$$\tau_2 = \left[\binom{n}{2} - 1\right]\frac{1}{2} + n\left[\frac{p/n}{q/\binom{n}{2} + p/n}\right] = \Theta(n^2).$$

The total expected wait time is thus

$$E[T_{\mathrm{rm}}] = \frac{n}{2}\left[\frac{\tau_1}{2} + \frac{\tau_2}{4}\right] = \Theta(n^3). \qquad \square$$

The above result implies that $E[T_{\mathrm{rm}}]/E[T_{\mathrm{r2}}]$ belongs to $\Theta(q)$. Hence, the memory-assisted version can indeed reduce the order of growth of the expected runtime. For example, if $q = 1/n^c$ for some positive constant c, then the ratio becomes $\Theta(1/n^c)$.

8 Memory Management

For practical considerations, memory consumption is an important issue. We remark that both RLS-m and RLS-m+ require only a very limited amount of memory. For RLS-m, only points within the 1-ball of the current point need to be stored. In other words, we need to store at most n points. For RLS-m+, points within the k ring need to be generated. To do this, we may use a pseudorandom number generator to generate a random permutation of the $\binom{n}{k}$ points in the k ring. This can be done, for example, by a linear congruent generator. What we need to remember is only the two parameters of the generator and its current output. The output number can then be mapped to a bit pattern. As an example, consider the case where $n = 4$. Suppose we are now exploring

the 2-ring. We may perform the following mapping:

$$1 \rightarrow 1100$$
$$2 \rightarrow 1010$$
$$3 \rightarrow 1001$$
$$4 \rightarrow 0110$$
$$5 \rightarrow 0101$$
$$6 \rightarrow 0011$$

This mapping can be generalized to arbitrary n and k in a straightforward way. Suppose we denote it by $f_{n,k}(x)$, where x is an integer satisfying $1 \leq x \leq \binom{n}{k}$. This mapping can be done in linear time. To show this, observe that for $n \geq 2$, the leftmost bit can be determined by comparing x with $\binom{n-1}{k-1}$. It is equal to 1 if $x \leq \binom{n-1}{k-1}$, and 0 otherwise. Next, we determine the other $n-1$ bits. If $x \leq \binom{n-1}{k-1}$, the remaining $n-1$ bits can be obtained by $f_{n\text{-}1,k\text{-}1}(x)$. Otherwise, they can be obtained by $f_{n\text{-}1,k}(x - \binom{n-1}{k-1})$. Since there are n bits in total, we need only n steps to compute $f_{n,k}(x)$. In other words, generating all points in the k ring in a pseudorandom way can be done efficiently in linear time without much memory. It is thus possible to implement RLS-m+ and apply it to an unknown landscape with no additional memory.

While the above method is suitable for RLS-m+, it cannot be applied to other memory-assisted algorithms. In general, we have to store all the visited points until a fitness improvement is made. Memory usage grows when an algorithm gets stuck in some local maxima, and the growth rate is proportional to the time that the algorithm is being trapped at that local maximum. In practice, the algorithm will not be executed indefinitely. If there is no fitness improvement over a long period of time, it should be stopped. Either the best solution obtained is produced as an output or another initial point is randomly chosen and the algorithm is restarted. In either case, the allocated memory will be released. Therefore, for practical implementation, we can set an upper limit on the number of visited points stored. We denote this maximum number by S, which typically is much smaller than the size of the problem space, 2^n. There are many different ways to store these S points. Some examples are listed below.

1 **Raw List.** The simplest one is to store them in a raw list. To check whether a newly generated point has been visited before, one needs to go through the entire list. The expected time is $O(L)$, where L is the expected length of the list.

2 **Sorted List.** If the raw list is sorted, then the search efficiency can be improved to $O(\log L)$. On the other hand, inserting a new element to the list takes $O(L)$ steps.

3 **Binary Tree.** A binary tree of depth n is maintained. To store a new point in the memory, a leaf node of the tree is created. To check whether a point has been visited before, one needs to traverse the tree. The complexity of this operation is $O(n)$.

4 **Hash Table.** A raw list is also maintained, but the search procedure is different. We create a hash table of P entries, with each entry consisting of $\lceil \log_2 S \rceil$ bits.

The table size, P, is typically chosen such that it is a little larger than S—say, 10 times larger (Mackay, 2004). A hash function $h : \{0, 1\}^n \to \{0, 1, \ldots, P-1\}$ is chosen. Given a point x, we compute a hash value, $i = h(x)$, which is used as an index to the hash table. There are two possibilities.

- If the ith entry in the table is empty, then $\boldsymbol{x}$ has not been visited before and it will be stored in the memory. Suppose that it is the sth point to be stored. The following two operations will be performed: (i) The ith entry of the hash table is updated to s; (ii) x will be appended to the end of the raw list, that is, the sth position.
- If the ith entry in the table is occupied and is equal to s, then examine the sth point in the raw list. If that point is equal to x, then x has been visited before. Otherwise, a collision in the hash table has occurred and rehashing should be performed. Another hash value is generated and the above procedure is repeated.

The time complexity of this method depends on the hashing function and rehashing method.

To better illustrate the gain that can be obtained by memory-assisted algorithms, we implement the (1+1) EA-m using two different memory management mechanisms: the raw list and the hash table. Using a raw list to store the points is simple to implement and efficient when the number of points to be stored is small. Using a hash table is more efficient when the number of points is large.

In our implementation, we use the simple division method for hashing:

$$h(\boldsymbol{x}) \triangleq \left(\sum_{i=1}^{n} 2^{n-i} x_i \right) \mod P,$$

where P is chosen to be a prime number. Furthermore, we adopt quadratic rehashing (e.g., Langsam et al., 1995): when there is a collision in the hash table, the new hash value is computed as $\left(h(\boldsymbol{x}) + j^2\right) \bmod P$, where j is the number of times that collisions have occurred.

In theory, P can be any prime number. On the other hand, the following result shows that for certain values of P, the hash function h has the following nice property.

THEOREM 18: *If P is a prime number greater than $2n$ and has a primitive root two, then $h(x) \neq h(y)$ for all x, y whose Hamming distance is one or two.*

PROOF: First consider the case where $d(x, y) = 1$. The statement is true because $(\pm 2^i) \not\equiv 0 \bmod P$. Next consider $d(x, y) = 2$. In that case, $(h(x) - h(y)) \bmod P$ must be in the form of $(\pm 2^i \pm 2^j) \bmod P$, for some i, j such that $0 \leq i < j \leq n$. We need to show that $2^i \pm 2^j \not\equiv 0 \bmod P$.

Suppose $2^i - 2^j \equiv 0 \bmod P$. This implies that $2^{j-i} \equiv 1 \bmod P$. Since 2 is a primitive root of P, the exponent $j - i$ must be an integer multiple of $P - 1$, which cannot be true since $j - i \leq n < P - 1$.

Suppose $2^i + 2^j \equiv 0 \bmod P$. This implies that $2^{j-i} \equiv -1 \bmod P$. Then $2^{2(j-i)} \equiv 1 \bmod P$. By the same argument as above, $2(j - i)$ must be an integer multiple of

$P-1$. In other words, $j-i$ is an integer multiple of $\frac{P-1}{2}$, which is impossible since $j-i \leq n \leq \frac{P-1}{2} < P-1$. □

If P is chosen so that the condition in Theorem 18 is satisfied, then there will not be any collision for points within the Hamming 1-ball of the current point. This can reduce the amount of rehash, since standard mutation often flips only one bit of the current point.

Based on the division method, $h(x)$ has a time complexity of $O(n)$. If y is obtained from x by flipping k bits, where the bit indices are in $\mathcal{F} \triangleq \{n_1, n_2, \ldots, n_k\}$, then

$$h(y) = h(x) + \sum_{i \in \mathcal{F}} 2^{n-i} \mod P.$$

Therefore, if $h(x)$ is known, the computation of $h(y)$ can be performed in $O(k)$ steps, where k is the number of bits flipped. Typically, when y is generated from x by the standard mutation, k is much smaller than n. If the size of the hash table is sufficiently larger than S, then the number of rehashing is small and the time complexity of the memory mechanism should also be small.

9 Empirical Studies

In this section, we compare the performance of (1+1) EA and (1+1) EA-m. For the (1+1) EA-m, we have two implementations. One is based on the raw list and the other is based on the hash table. When the hash table is used, we always choose P to be the smallest prime number with primitive root two that is greater than $10S$. This is done by precomputing a list of prime numbers with primitive root two. All algorithms are implemented in MATLAB. All experiments are performed on Dell Optiplex GX745, which has 1 GB RAM and an Intel(R) Core(TM)2 CPU with clock frequency 2.13 GHz.

First, we run (1+1) EA and the two implementations of (1+1) EA-m over three test functions which we have discussed in the previous sections. They are OneMax, LeadingCouples, and Jump. Since memory-assisted algorithms are mainly designed for problems with expensive fitness evaluations, in our experiment, we artificially add computation time to evaluate fitness for the above three test functions. For each evaluation, we add $10n$ μs to the actual evaluation time. This simulates the situation where the fitness evaluation time grows linearly with n. It is easy to imagine that the larger the value of the extra time added, the more improvement the (1+1) EA-m can make.

For each test function, the average runtime and the average CPU time of the two algorithms are plotted, based on 1000 simulation runs. Recall that runtime refers to the number of fitness evaluations performed. Hence, the two implementations of (1+1) EA-m have the same average runtimes. The CPU time is measured in terms of seconds. For each test function, we also plot a box-and-whisker plot for a particular value of n, using the built-in function boxplot in MATLAB. In each plot, there are three boxes that correspond to (1+1) EA and the two implementations of (1+1) EA-m. The boxes have lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the boxes to show the extent of the rest of the data. Outliers are data with values beyond the ends of the whiskers.

Figure 2 shows the average result for OneMax. For the hashing method, we set $S := 6n$, since OneMax is a unimodal function and the expected number of points to be stored in the memory is equal to en. From the figure, we can see that both versions

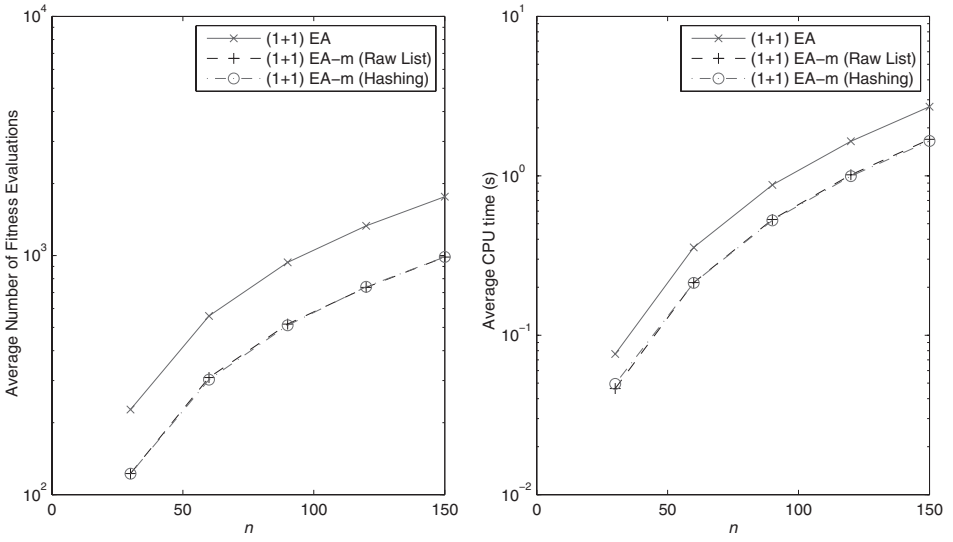

Figure 2: Average number of fitness evaluations and average CPU time for OneMax.

of (1+1) EA-m outperforms (1+1) EA. The reason is that the time used for maintaining the memory is less than the time used for evaluating the fitness of visited points. If we take a closer examination at the points for $n = 150$, we find that (1+1) EA-m saves around 44% of runtime and around 38% of CPU time. The saving of CPU time is less than 44% because the memory mechanism consumes part of the CPU time. Figure 3 shows the box-and-whisker plot for the case where $n = 150$. It can be seen that the two versions of (1+1) EA-m have similar performance and both of them outperform (1+1) EA significantly.

Figure 4 shows the average result for LeadingCouples. For the hashing method, we set $S := n^2$. From the figure, we can observe the same phenomenon as in OneMax, when we compare (1+1) EA with (1+1) EA-m with hashing. However, the average CPU time of (1+1) EA-m with raw list increases quickly with n. The reason is that for LeadingCouples, the number of visited points are larger and thus the search time for sequential search increases quickly. Figure 5 shows the box-and-whisker plot for the case where $n = 50$. We can see that (1+1) EA-m with hashing has the best performance.

Figure 6 shows the result for $\text{Jump}_{\alpha n,n}$, where $\alpha = 1/3$. For the hashing method, we set $S := 2\binom{n}{n/3}$. This value for S is quite large and is proportional to the number of points in the large gap between a local maximum and the global maximum. We can see that the computation times of all three algorithms grow exponentially, but (1+1) EA has a much steeper slope. The hashing method is more efficient for large n, just as in the case for OneMax and LeadingCouples. Similar behavior can also be observed in Figure 7, which shows the box-and-whisker plot for the case where $n = 15$.

In all three test functions, we have artificially added computation time for evaluating fitness. It is quite clear that if fitness evaluation is more expensive than memory management, the memory-assisted approach will benefit. To further demonstrate this fact, we consider a fitness function that is intrinsically expensive. It is an instance of the well known Max-3Sat problem and is presented in Papadimitriou (1994). In total, there

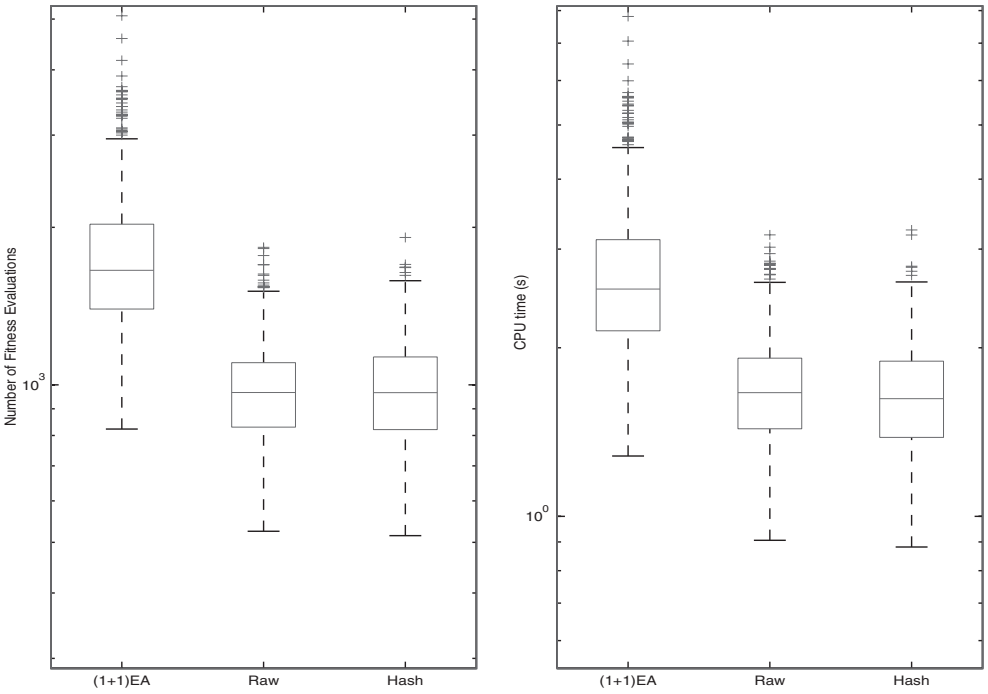

Figure 3: The box-and-whisker plot for ONEMAX with $n = 150$.

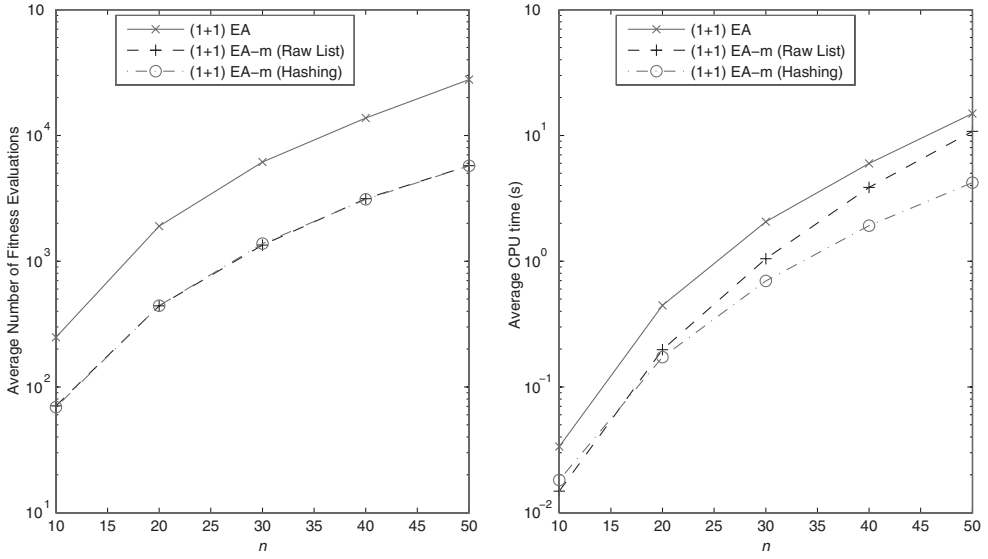

Figure 4: Average number of fitness evaluations and average CPU time for LEADING-COUPLES.

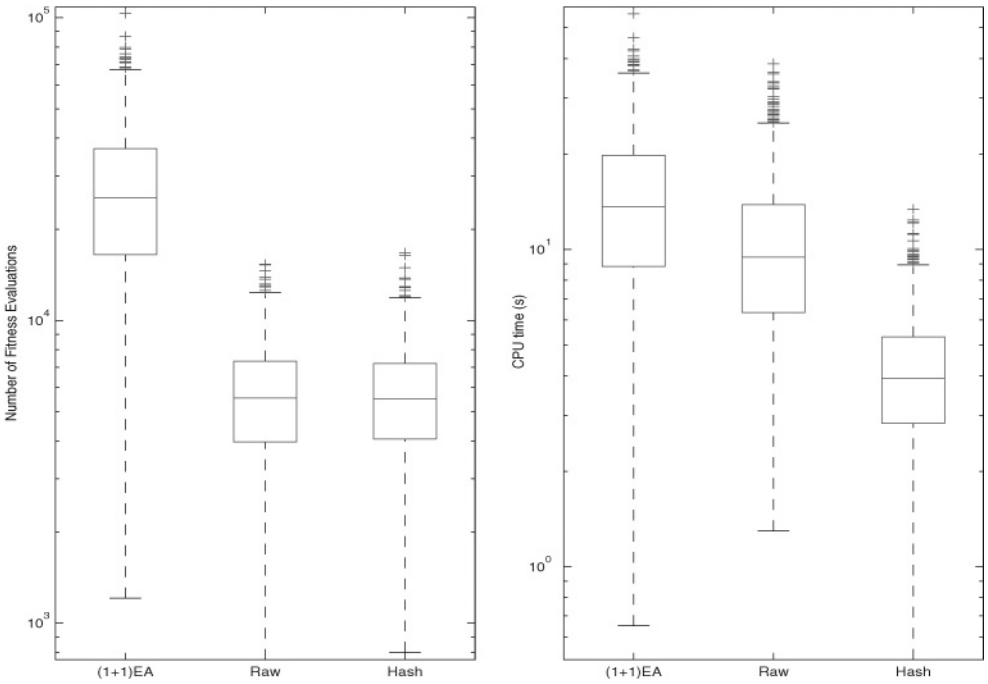

Figure 5: The box-and-whisker plot for LEADINGCOUPLES with $n = 50$.

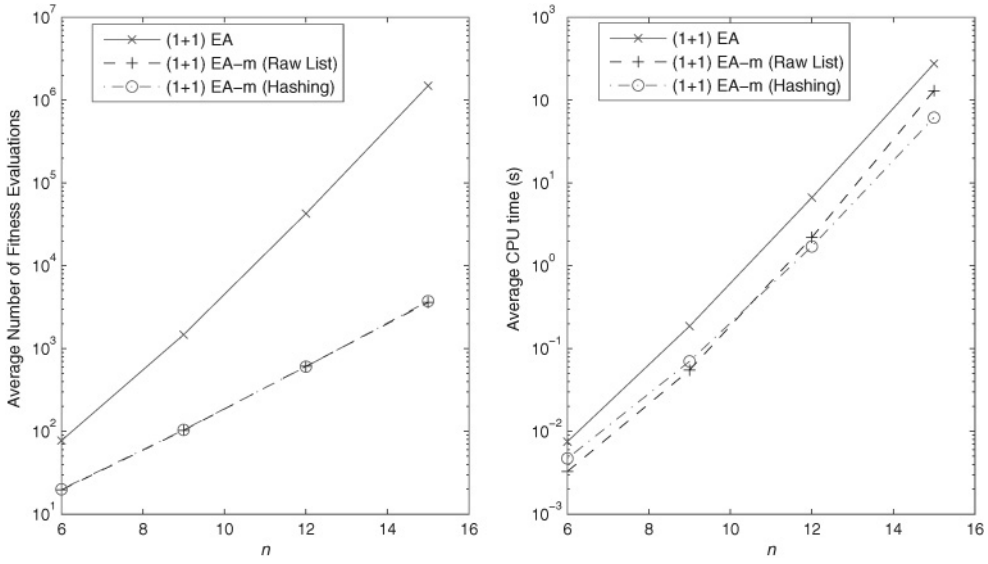

Figure 6: Average number of fitness evaluations and average CPU time for JUMP.

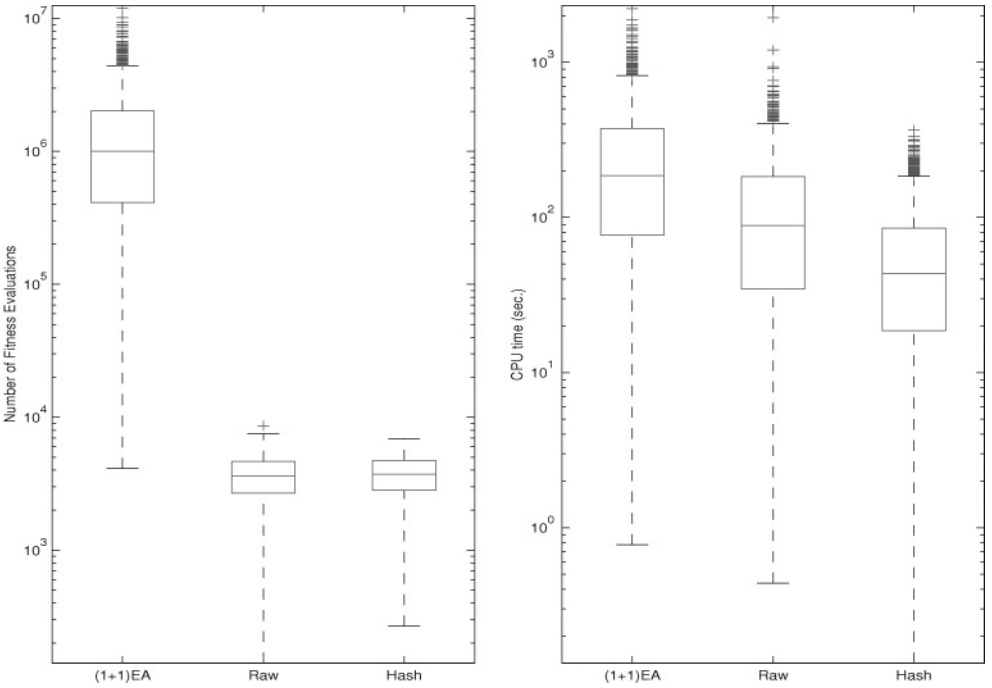

Figure 7: The box-and-whisker plot for JUMP with $n = 15$.

are n clauses of length 1 and $n(n-1)(n-2)$ clauses of length 3, which are specified below:

- $x_i, i = 1, 2, \ldots, n$, and
- $x_i \vee \bar{x}_j \vee \bar{x}_k, (i, j, k) \in \{1, 2, \ldots, n\}^3$, where $i \neq j \neq k \neq i$.

By inspection, the optimal point is the all-one vector, $\mathbf{1}$, because it satisfies all the clauses. Nevertheless, it is shown in Droste et al. (2002b) that this problem instance cannot be optimized efficiently by any search heuristic that fulfills a reasonable hypothesis.

To put this problem into our framework, we express it in terms of the following fitness function (Droste et al., 2002b):

$$\textsc{Count}_n(\boldsymbol{x}) = \sum_i x_i + \sum_i \sum_{j \neq i} \sum_{k \neq i,j} (1 - (1 - x_i)x_j x_k).$$

Note that the time complexity of computing the fitness value for any given x is $O(n^3)$, which is intrinsically expensive. Therefore, in our experiment, we do *not* add any extra computation time for fitness evaluation.

Figure 8 shows the result for COUNT. Since the runtime for this problem increases rapidly with n, we only conduct experiments for n up to eight. Since n is small, we set $S := 2^n$ for the hashing method, so that the hash table will not overflow. From the figure, we can see that both versions of (1+1) EA-m outperform (1+1) EA significantly

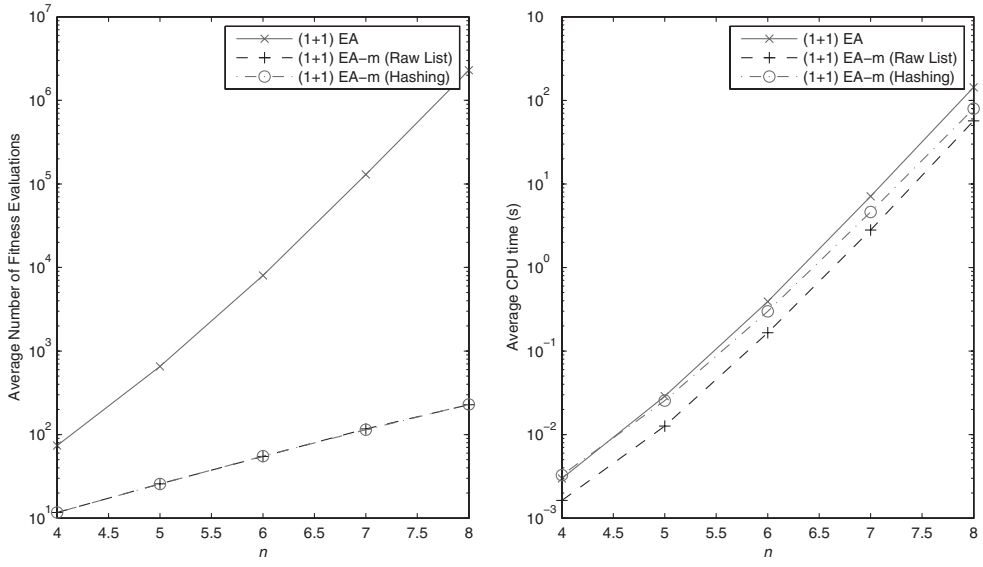

Figure 8: Average number of fitness evaluations and average CPU time for COUNT.

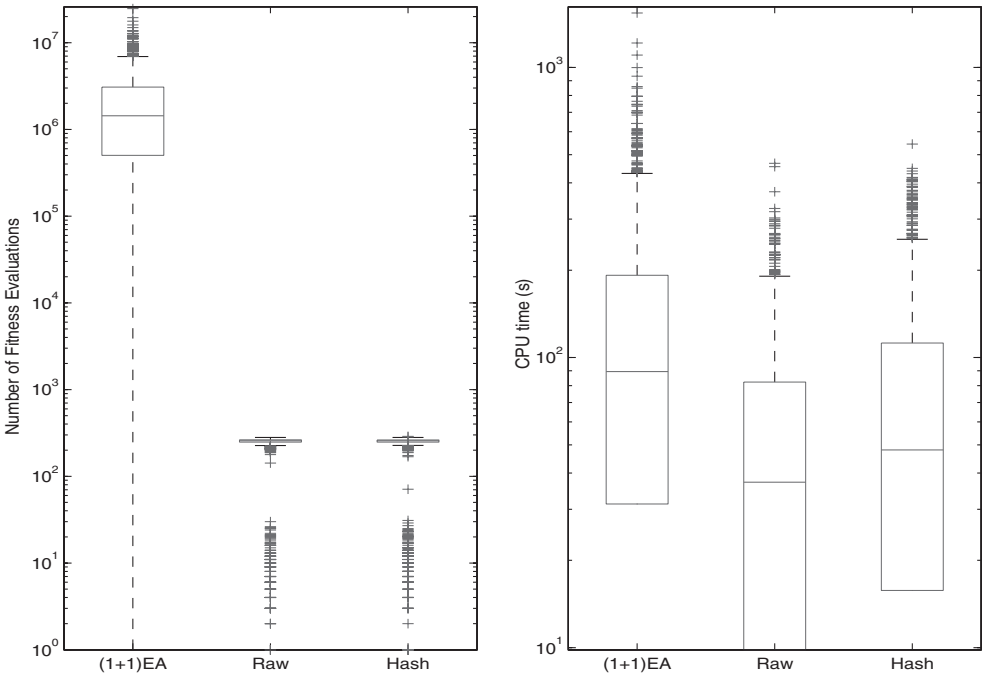

Figure 9: The box-and-whisker plot for COUNT with $n = 8$.

in terms of runtimes. They also outperform (1+1) EA in terms of CPU time. More saving is obtained with the use of the raw list because n is small.

One may argue that the improvement in CPU time is not that much. This is true, because in this experiment, no extra time is added in evaluating fitness. As the problem size is small ($n \leq 8$), the actual fitness evaluation time is not very long, and therefore, the improvement is not significant. On the other hand, the difference between (1+1) EA and (1+1) EA-m (with hashing) increases with n increased. The reason is that the fitness evaluation time for COUNT increases with n^3. More significant improvement is expected for large n.

The box-and-whisker plot for the case where n is equal to eight is shown in Figure 9. It can be seen that the median and quartile values achieved by the two versions of (1+1) EA-m are smaller than that achieved by (1+1) EA.

10 Conclusions

Motivated by the potential performance gain of memory-assisted algorithms, we investigate the effect of memory on expected runtime for pseudo-Boolean function optimization. To facilitate our analysis, we propose two memory-assisted algorithms, (1+1) EA-m (with raw list or hash table option) and RLS-m+ (with RLS-m variant if the function is a priori known to be unimodal). Their expected runtimes on some pseudo-Boolean functions have been analytically obtained and compared with their counterparts where memory is not used. For unimodal functions, with the use of memory, the expected runtime of RLS can be reduced, but by at most one half. We then generalize our result and construct a unified framework that works on both RLS and (1+1) EA. It is shown that in general, all functions can be treated as unimodal functions and analyzed accordingly using a new concept known as spatially invariant neighborhood. Under this framework, it is shown that the use of memory is promising for multimodal functions. For functions having gaps, the order of growth of the expected runtime of (1+1) EA can be significantly reduced. A lower bound of the expected runtime for a class of (1+1) EAs is derived. An example using the function LEADINGCOUPLES is used to demonstrate that a simple memory-assisted algorithm can reduce the order of growth of the expected runtime. It is also shown that for some functions, the speedup in expected runtime can be from exponential to polynomial; a simple example of this is the ONEMAX-ZEROSPIKE function. The results suggest that the use of memory assistance is more beneficial for hard functions than for easy functions, if we do not consider fitness evaluation costs.

On the other hand, there are many practical applications involving expensive and/or time-consuming fitness evaluations costs. For such applications, the use of memory assistance will without question be beneficial for both easy and hard problems. We note that for some practical applications, even a 50% reduction in computation time can be substantial—imagine a fitness evaluation involving Monte Carlo simulation that may take hours to converge. For many fitness functions, the potential benefit can be much higher. Our results indicate that a memory-assisted approach is promising, especially for applications whose fitness evaluation is time-consuming and expensive. To substantiate this claim, we have performed an empirical study. The (1+1) EA-m is implemented based on the raw list and hash table. Numerical examples show that these versions have shorter actual CPU runtimes than their memoryless counterpart, provided that the fitness evaluation takes a reasonably long time. In our simulation, we assume that the fitness evaluation time increases linearly with problem size.

In the past two decades, local search methods have been widely studied and developed. It was stated in Aardal et al. (1997) that "local search has reinforced its position as a standard approach in combinatorial optimization." However, local search, in its pure form or its randomized variant, is well known to have the shortcoming of the possibility of getting trapped in local maxima. Our proposed version, RLS-m+, with a negligible amount of memory overhead, can completely overcome this limitation. We believe that RLS, used either alone or as a component in a more sophisticated search algorithm, should be replaced by RLS-m+ for practical applications.

Nowadays, evolutionary algorithms are frequently employed to optimize black box functions (Wegener, 2005) of which we have no or little knowledge. Nonetheless, almost all researchers and industry practitioners will agree that domain knowledge, given or acquired, about the problem will help the search. Such knowledge may help change the search strategies online or help develop better heuristic algorithms. In this connection, RLS-m+ will naturally measure the largest gap that it encounters during the search. This seems to be a piece of useful knowledge about the search landscape that may be used to benefit other search methods. Thus, after applying RLS-m+, we at least learn something about the landscape—the minimum size of the largest Hamming gap in the landscape.

To our knowledge, this is the first paper that rigorously analyzes theoretically the impact of memory assistance on an evolutionary algorithm. Admittedly, our analysis has only been done on two very simple evolutionary algorithms: (1+1) EA and RLS; and memory assistance is provided only up to the point of enabling a local improvement. In tabu search and nonrevisiting stochastic search (Yuen and Chow, 2009), more advanced use of memory is made. It would be worthwhile to extend our analysis to these more advanced algorithms in the future.

From an algorithmic design viewpoint, two new algorithms, that is, (1+1) EA-m (with raw list or hash table option) and RLS-m+, have been proposed. We believe that they have good potential to be used as one of the partner algorithm within a memetic algorithm. The RLS-m+ algorithm is especially promising, since it requires no memory overhead and provides a natural and systematic random local search that flips multiple bits; it extends the impractical RLS to a much more practical version. Moreover, as mentioned above, it is a natural gap measurer for an unknown problem landscape.

One existing problem with the (1+1) EA-m is that for hard multimodal problems, for example, a function with large gaps, to find a point that improves will take a long time. Thus the memory may be full and needs to discard points. This problem will need to be addressed by a better design in the future.

Acknowledgments

We would like to thank the anonymous reviewers for their detailed and constructive comments, which led to a substantial improvement of this paper. The first author is grateful to Mr. Bruce Tong for his suggestions on the design of the hash table, and to Dr. Kenneth W. Shum for discussions on the proof of Theorem 18.

References

Aardal, K., S. Hoesel, J. K. L., and Stougie, L. (1997). A decade of combinatorial optimization. *CWI Tracts 122*, pp. 5–14.

Chow, C. K., and Yuen, S. Y. (2008). A non-revisiting particle swarm optimization. *Proceedings of CEC*, pp. 1879–1885.

Dietzfelbinger, M., Naudts, B., Hoyweghen, C. V., and Wegener, I. (2003). The analysis of a recombinative hill-climber on H-IFF. *IEEE Transactions on Evolutionary Computation*, 7(5):417–423.

Droste, S., Jansen, T., and Wegener, I. (2002a). On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276(1–2):51–81.

Droste, S., Jansen, T., and Wegener, I. (2002b). Optimization with randomized search heuristics—The (A)NFL theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131–144.

Fong, K. F., Hanby, V. I., and Chow, T. T. (2008). A robust evolutionary algorithm for HVAC engineering optimization. *HVAC&R Research*, 14(5):683–705.

Fong, K. F., Yuen, S. Y., Chow, C. K., and Leung, S. W. (2010). Energy management and design of centralized air-conditioning systems through the non-revisiting strategy for heuristic optimization methods. *Applied Energy*, 87(11):3494–3506.

Garnier, J., and Kallel, L. (2000). Statistical distribution of the convergence time of evolutionary algorithms for long path problems. *IEEE Transactions on Evolutionary Computation*, 4(1):16–30.

Garnier, J., Kallel, L., and Schoenauer, M. (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation*, 7(1):167–203.

Glover, F., and Laguna, M. (1997). *Tabu search*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329.

He, J., and Yao, X. (2002). From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511.

He, J., and Yao, X. (2003). Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artificial Intelligence*, 145(1–2):59–97.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Jansen, T., Jong, K. A. D., and Wegener, I. (2005). On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13(4):413–440.

Jansen, T., and Wegener, I. (2002a). The analysis of evolutionary algorithms—A proof that crossover really can help. *Algorithmica*, 34(1):47–66.

Jansen, T., and Wegener, I. (2002b). Evolutionary algorithms—How to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5(6):589–599.

Jansen, T., and Wegener, I. (2007). A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-Boolean functions of unitation. *Theoretical Computer Science*, 386(1–2):73–93.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Langsam, Y., Augenstein, M. J., and Tenenbaum, A. M. (1995). *Data structures using C and C++*, 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Mackay, D. J. C. (2004). *Information theory, inference, and learning algorithms*. Cambridge, UK: Cambridge University Press.

Mühlenbein, H. (1992). How genetic algorithms really work: Mutation and hillclimbing. *Proceedings of the PPSN*, pp. 15–26.

Neumann, F. (2008). Expected runtimes of evolutionary algorithms for the Eulerian cycle problem. *Computers & Operations Research*, 35(9):2750–2759.

Neumann, F., and Wegener, I. (2007). Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40.

Oliveto, P. S., He, J., and Yao, X. (2007). Computational complexity analysis of evolutionary algorithms for combinatorial optimization: A decade of results. *International Journal of Automation and Computing*, 4(3):281–293.

Oliveto, P. S., He, J., and Yao, X. (2008). Analysis of population-based evolutionary algorithms for the vertex cover problem. *Proceedings of the CEC*, pp. 1563–1570.

Papadimitriou, C. H. (1994). *Computational complexity*. Reading, MA: Addison-Wesley.

Rudolph, G. (1996). How mutation and selection solve long path problems in polynomial expected time. *Evolutionary Computation*, 42(2):195–205.

Rudolph, G. (1997). *Convergence properties of evolutionary algorithms*. Hamburg: Verlag Dr. Kovač.

Srinivas, M., and Patnaik, L. M. (1996). On modeling genetic algorithms for functions of unitation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(6):809–821.

Sudholt, D. (2005). Crossover is provably essential for the ising model on trees. *Proceedings of the GECCO*, pp. 1161–1167.

Sudholt, D. (2006). On the analysis of the (1+1) memetic algorithm. *Proceedings of the GECCO*, pp. 493–500.

Wegener, I. (2005). *Complexity theory*. Berlin: Springer-Verlag.

Witt, C. (2006). Runtime analysis of the (μ+1) EA on simple pseudo-Boolean functions. *Evolutionary Computation*, 14(1):65–86.

Yuen, S. Y., and Chow, C. K. (2008a). Applying non-revisiting genetic algorithm to traveling salesman problem. *Proceedings of the CEC*, pp. 2217–2224.

Yuen, S. Y., and Chow, C. K. (2008b). A non-revisiting simulated annealing algorithm. *Proceedings of the CEC*, pp. 1886–1892.

Yuen, S. Y., and Chow, C. K. (2009). A genetic algorithm that adaptively mutates and never revisits. *IEEE Transactions on Evolutionary Computation*, 13(2):454–472.